Defence Research and Development Canada
Recherche et développement pour la défense Canada

Autonomous Sustain and Resupply, Phase 1 Report

R. Vaughan, Y. Litus, J. Wawerla, and A. Lein
Simon Fraser University

Contract Scientific Authority: G. Broten, DRDC Suffield

Defence R&D Canada

Contract Report

DRDC Suffield CR 2009-067

October 2008

Autonomous Sustain and Resupply, Phase 1 Report

R. Vaughan, Y. Litus, J. Wawerla, and A. Lein
Simon Fraser University
School of Computing Science, Autonomy Lab
Burnaby BC V5A 4S6

Contract Number: W7702-07R146-001

Contract Scientific Authority: G. Broten (403-544-4732)

Defence R&D Canada – Suffield

Contract Report

DRDC Suffield CR 2009-067

October 2008

Autonomous Sustain and Resupply Phase 1 Report

Submitted to DRDC in support of Contract W7702-07R146/001/EDM

Richard Vaughan
Simon Fraser University

March 31, 2008

Contents

Chapter 1

Introduction, Approach, and Progress Overview

1.1 Introduction

This report presents our progress to date on the DRDC Autonomous Sustain and Resupply project. We are using the ASR problem as the key motivating problem for the AI and autonomy research in our lab.

The document is structured as follows. First we state our general approach, which is unchanged from the proposal. Then we summarize our progress to date, referencing detailed descriptions of our work provided as chapters of this report. Next we provide the research plan, which is slightly revised from the version in our proposal. Next is the literature review, followed by our formal model of the problem, and then a design for a planning-based solution. The remainder is a series of chapters describing our progress during Phase 1 on various aspects of the ASR problem.

1.2 Approach

Considering the target problem, we can identify three main sub-problems that are conventionally considered in isolation. These are:

1. Multi-robot task allocation, preferably distributed.
2. Point-to-point path-planning and navigation in dynamic environments with uncertainty.
3. Individual robot self-maintenance for long-term operation.

Each of these components is a mature research area in itself, with a wealth of literature. For example, a definitive review of multi-robot task allocation methods is given in [32]. Point-to-point navigation is extremely well-studied; for example a current DARPA project addresses the problem of fast and robust navigation in outdoor unstructured environments. Each problem has well-known approaches that could be applied to the target task.

We believe that the novelty and research value of this project is in the integration of these components. This provides two research questions: (i) what are the precise boundaries and interfaces between the components, and what alternative approaches are suitable to be combined in this way? (ii) what synergistic effects can be found between these components, so that new, task-specific methods can be designed to solve the task more efficiently than by a naive combination of previous approaches?

To reflect this view, we are following the following methodology, as described in our proposal. First we analyze the control task formally and review the relevant literature on the component tasks. Then we engineer a "classical" system: a modular integrated system using the most appropriate methods from the literature. This engineering exercise has two benefits; it satisfies the requirements of the project by producing a working system, thus minimizing the research risk of the project, and it gives us essential experience and insight into a running system. Our preference is for decentralized, distributed methods wherever possible, for their advantages in robustness through redundancy, and scalability.

Next, we attempt to deconstruct the classical system, creating novel, problem-specific techniques to create a "fast and frugal" system. Using the insights gained from our the classical approach, and many of the system component already built, we will attempt to devise task-specific heuristics that provide increases in performance and/or scalability, and/or system cost and complexity.

Throughout, we perform principled evaluations of the system performance, and attempt to identify the cost/benefit trade-offs in our design decisions.

1.3 Progress in this period

1.3.1 Personnel

We have employed a team students from the Autonomy Lab to work on the project. Two senior Phd students, Jens Wawerla and Yaroslav Litus, and one senior MSc student, Adam Lein, have been employed to date. The vast majority of the budget is allocated to paying student salaries. All these students contributed to this report.

1.3.2 Literature review

We have reviewed the state of the art in solving the target problem. The key references are described in Chapter 4, which is a high-level review. More specific reviews of sub-topics can be found at the beginning of each of the research chapters at the end of the report.

1.3.3 Modeling Adaptive Sustain and Resupply

We have developed a series of formal models of the ASR problem, and Chapter 3 provides the latest iteration of these.

1.3.4 Distributed solutions to gradient optimization problems

Robot rendezvous is a vital part of the ASR problem, and we have been developing scalable methods of achieving multi-robot, multi-place rendezvous using fast, distributed algorithms. We have made considerable progress in the last four months, and Chapter 6 describes our approach in detail.

1.3.5 Optimal robot recharging strategies

ASR requires robots to manage their energy supplies so as to provide continuous operation. We are investigating the space of optimal recharging strategies, and have some interesting an novel results described in Chapter 7.

1.3.6 Adaptive Foraging

We are investigating a family of partial ASR solutions based on models of foraging behaviour in animals. Chapter 8 details our progress in this period, in which we have reproduced and improved upon the previously most successful territorial foraging methods.

1.3.7 Simulation platform

We have worked extensively on the Stage multi-robot simulator, making it considerably more useful for this project. The aims of the current development are:

1. Add an abstract resource generation, transfer and consuming model to Stage, to model the movement of energy and other resources between robots.
2. Add a matching generic resource interface to the Player server.
3. Develop Stage models that approximate the target robot platform, to be discussed with DRDC. The simulation experiments will be performed with a mixture of the DRDC target platform and models of our locally available robots.
4. Performance and scaling enhancements to ensure that Stage easily supports the target number of robot models. Our internal target is 20 complex (i.e. laser and/or camera equipped, with wireless communication model) robots in real time on standard PC hardware in 90% of real-time or better, and 200 robots in usable but slower-than-real-time mode. Key performance gains will be obtained by improving the raytracing subsystem, and possibly by exploiting SIMD computation on GPU hardware. Minor performance gains will be obtained by profiling and refactoring frequently-called code.
5. Ongoing bugfixes, documentation and user support throughout the life of the program.

To date we have made good progress on point 1. Figure 1.1 shows a screenshot of Stage running a resource-transport demonstartion shown during our visit to DRDC in March 2008. The abstract resources are shown as coloured “jewels” and are generated, exchanged, and consumed by various simulation models. The red mobile robots are autonomously transporting jewels from the green “source” to the red “sink” location.

We have greatly exceeded our expectations on point 3 - multi-robot performance. See Chapter 9 for a detailed description of the progress on improving Stage’s scalability.

In addition, we have added a new feature to Stage: plug-in control modules that allow users to attach arbitrary compiled programs to models. Typically this programs are used to control robots, but they can also be used to run automated experiments, logging, visualization, etc. This work is in an early experimental form now, but it is functional: the transportation demo described above uses it for robot controllers. It will be reported in detail in future.

Parallel to this project (and funded externally) we have continued our efforts with education and promotion of the Player, Stage and Gazebo platforms in the global research community.

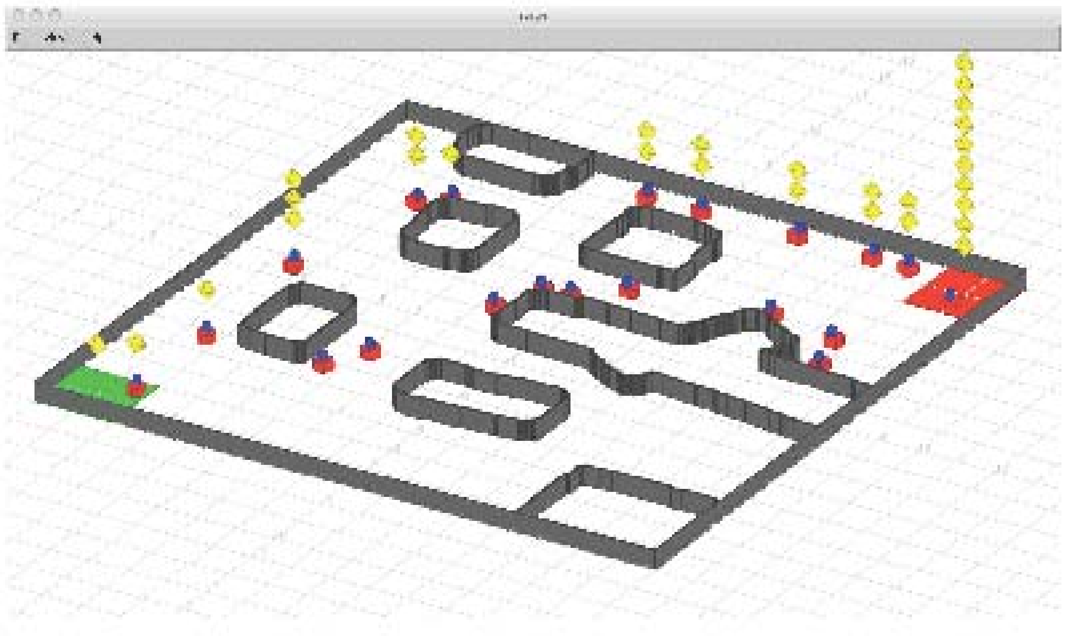


Figure 1.1: Screenshot from the current development version of Stage, showing asbtract resource model. The red robots are transporting units of resource (yellow 'jewels') from the source location to the goal location. All code and configuration files are in Stage CVS HEAD.

1.3.8 Real robot platform

Our custom-designed Chatterbox robot prototype is complete see Figure 1.2 and we are in a testing phase. Once the robot has been shown to be reliable, we will manufacture 30-40 robots, depending on the final cost. Robot construction is not funded by this project, though consumables such as batteries are funded. The complete population will be ready for use probably by the end of summer 2008, and in case by Christmas 2008, in plenty of time for Phase 2 real-robot experiments and demonstrations.

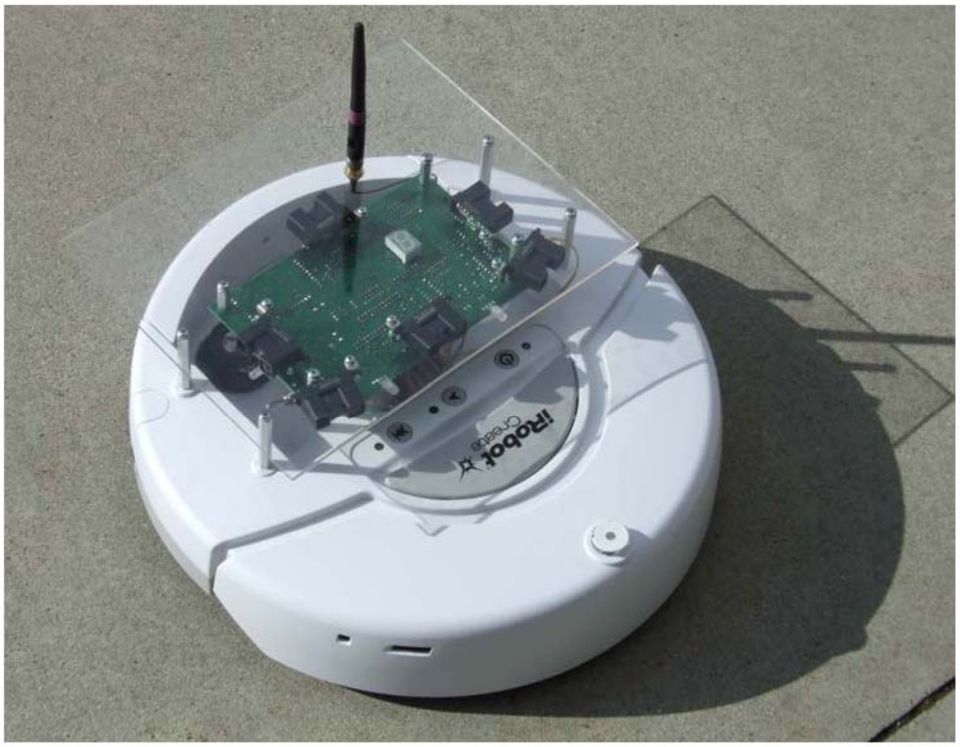


Figure 1.2: Autonomy Lab Chatterbox robot pre-production prototype.

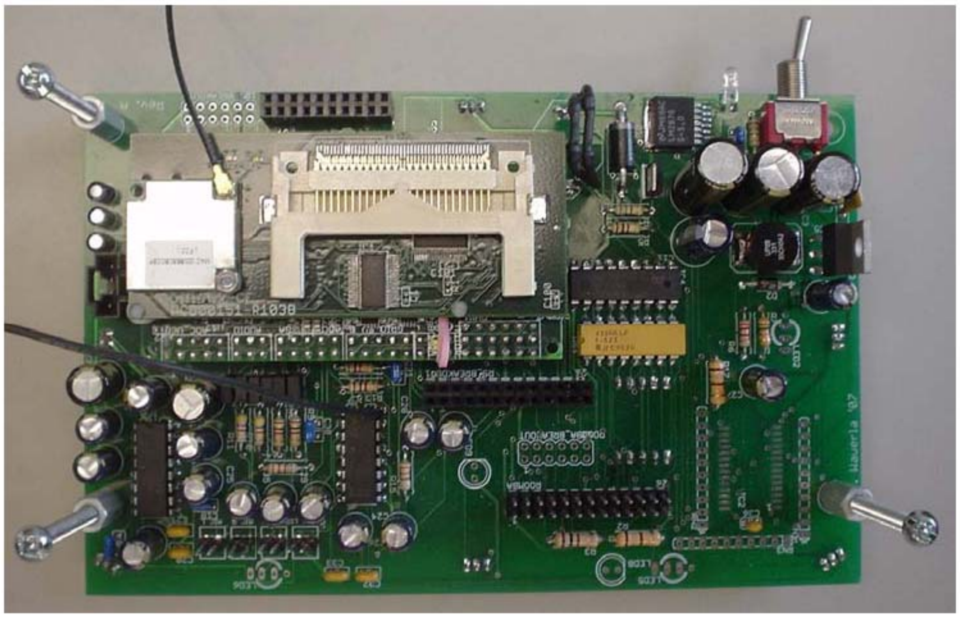


Figure 1.3: Autonomy Lab Chatterbox robot integration board pre-production prototype.

Chapter 2

Research Plan

2.1 Schedule and milestones

Our research plan is modified only slightly from our proposal, with no financial or deliverable changes, and as can be seen from the progress report, we are currently on schedule.

Our effort is structured as 5 complementary tasks. After the initial review and formalization exercise, tasks 2 and 3 are sequential, while tasks 4 and 5 run in parallel with the others.

1. Review of State of the Art, formal modeling, and design of research plan **(complete)**
2. Engineering a "classical" system that performs the task, including complete robot controller implementation and evaluation experiments.
3. Designing and engineering a "fast and frugal" solution, including complete robot controller implementation and evaluation experiments.
4. Infrastructure software development: Player and Stage modifications to support the project
5. Local project management and DRDC interaction

Each task is described in the sections below. Table 2.1 shows the proposed schedule of milestones and deliverables, by task. A deliverable of "source, docs, demo" indicates the provision to DRDC buildable source code, documentation and examples, and a runnable demonstration. In addition to these milestones and deliverables, progress reports will be delivered as required by DRDC, and academic papers published at the earliest opportunity.

Task 1: Review of State of the Art, problem formalization, and design of research plan

Completed.

Task	*Description*	*Deliverable*	*Due Mo/Yr*
Phase 1	**Research State of the Art**		
1.A	Report on state of the art		3/08
1.B	Revised Technical Approach	report	3/08
Phase 2	**Simulation**		
	Player/Stage development		
4.A	Stage resource model		6/08
4.B	Player resource interface	Source, docs, demo	6/08
	Classical system		
2.A	Single-robot resupply & recharge		7/08
2.B	Multi-robot (5) resupply & recharge		8/08
2.C	Multi-robot (50) resupply & recharge	Source, docs, demo	9/08
2.D	Real-robot reality check		
	CHECKPOINT 1		**9/08**
	Fast and Frugal system		
3.A	Multi-robot (5) resupply & recharge		2/09
3.B	Multi-robot (50) resupply & recharge	Source, docs, demo	3/09
3.C	Real-robot reality check		
	Evaluation		
3.D	Simulation evaluation	report	3/09
	CHECKPOINT 2		**3/09**
Phase 3	**Real Robots**		
	Classical System		
3.E	Multi-robot (5) resupply & recharge		7/09
3.F	Multi-robot (20) resupply & recharge		8/09
	CHECKPOINT 3		**9/09**
	Fast and Frugal system		
3.G	Multi-robot (5) resupply & recharge		11/09
3.H	Multi-robot (20) resupply & recharge	Source, docs, demo	1/10
	Evaluation		
3.I	Real-world evaluation		2/10
3.J	**FINAL REPORT**	report	**3/10**

Table 2.1: Schedule of tasks and deliverables. This report ends Phase 1 and we move to Phase 2

Task 2: "Classical" robot system

All personnel will contribute to the engineering and testing of a multi-robot system that performs the target task. The system is composed of components described in the planning review above, with the following requirements:

1. **Task Allocation:** a subsystem that dynamically allocates incoming job requests to robots in an efficient manner.
2. **Transportation:** once given a resupply task, a robot must rendezvous with the source of supplies, transfer supplies, then rendezvous with the sink of supplies. The environment is assumed to be at least partially unmapped, dynamic and hazardous. Robots are unreliable and may not complete a rendezvous, abandoning their task.
3. **Individual consumable management:** each robot must maintain its own supply of consumables to ensure long-duration operations. The basic practical version of this is that each robot must ensure that its battery (or fuel) never runs out, by visiting a recharger (or fuel station/tanker). If indefinite operation is required, recharging must occasionally occur, temporarily taking priority over other tasks.

We have previously built and published descriptions of working multi-robot systems that perform point-to-point transportation and recharging, as outlined in Chapter 4.

Robot controller implementation and evaluation

Control strategies devised in the project will be built into robot controllers, implemented to target the Player robot server. Our approach is to decouple the strategic decision making from the details of low-level navigation (except where any low-level information can be directly exploited, as happens for example in synergistic ant trail following). Examples of basis behaviours, to be used as building-blocks for high level planning include *go-to-location, rendezvous-with-other-robot, explore-to-recover-communications-link, find-nearest-recharger.*

Evaluation will be based on a series of simulation experiments (in Phase 2) and real-robot experiments (in Phase 3), and will use the metrics described above. We aim to demonstrate first a single resupply robot, then 5, then 50 robots. The large population will test the scalabilty of our approach.

Task 3: "Fast and Frugal" robot system

All personnel will contribute to the design and engineering of revised robot controllers. In this task we build on the classical system, and attempt to improve its performance, reliability, scalability and cost. We have previously been successful at creating very efficient, high-performance multi-robot systems, essentially by considering the target problem at two levels - a formal optimization problem, and a real-time robot behaviour generation problem - simultaneously. Essentially our approach is to always remember that the goal of the optimization system is eventually to make the robots move to achieve work. We seek to get the robots working early and fast, by pushing computation as far as possible down to individual robots, without centralized control, and by exploiting local sensing as much as possible to avoid the expense of maintaining large internal representations. This is essentially the application of Brooks' "the world is its own best model" advice, or the related principle of "world-embedded computation" described by Payton.

A thorough analysis of target task, including experience with an implemented system is required before a detailed plan for this task is possible. This is the area of highest risk, but potentially greatest scientific contribution in this proposal.

Task 4: Player and Stage software infrastructure development

Player and Stage require some minor extensions for the proposed research and experiments. These include:

- Add an abstract resource generation, transfer and consuming model to Stage, to model the movement of energy and other resources between robots.
- Add a matching generic resource interface to the Player server.
- Develop Stage models that approximate the target robot platform, to be discussed with DRDC. The simulation experiments will be performed with a mixture of the DRDC target platform and models of our locally available robots.
- Performance and scaling enhancements to ensure that Stage easily supports the target number of robot models. Our internal target is 20 complex (i.e. laser and/or camera equipped, with wireless communication model) robots in real time on standard PC hardware in 90% of real-time or better, and 200 robots in usable but slower-than-real-time mode. Key performance gains will be obtained by improving the raytracing subsystem, and possibly by exploiting SIMD computation on GPU hardware. Minor performance gains will be obtained by profiling and refactoring frequently-called code.
- Ongoing bugfixes, documentation and user support throughout the life of the program.

As a crucial dependency for other work, this task began immediately on project start, and will be largely complete by 9/2008, though ongoing maintenance and improvements will be necessary. All personnel will work on this task.

Parallel to this project (and funded externally) we will continue our efforts with education and promotion of the Player, Stage and Gazebo platforms in the global research community.

Task 5: Local project management and DRDC interaction

The project management plan is described in Section 2 of the proposal and is unchanged.

2.1.1 Checkpoints

In addition to the schedule of tasks and deliverables, we have marked three intermediate checkpoints, at 1 year, 18 months and 2 years. At each checkpoint we will perform a review exercise to assess if we are on track to reach the project goals, to identify and address any issues that are discovered, and to evaluate our work in the wider research context. The checkpoints are intended to be an internal sanity check and have no deliverables.

2.2 Deliverables

The main deliverables will be:

1. A report on the state of the art, and a revised work plan and Statement of Work, by 31/3/2008.
2. A live or video demonstration of a software system that performs the target task in a standard robot simulation, plus all source code, documentation, simulation configurations and supplementary files, by 31/3/09.
3. A live or video demonstration of a hardware and software system that performs the target task on real robots that model DRDC's target robots, plus all source code, documentation and supplementary files, by 31/3/10.
4. A final report including technical descriptions and critical evaluations of all components, by 31/3/10.

We will also deliver Additional demonstrations identified in table 1.1, along with the code, models, etc. used to perform each demo.

2.2.1 Requirements for software deliverables

All robot control software deliverables will:

1. target the well-known, Open Source, Player robot control platform. Player is the most-used platform in robotics research and education.
2. be written in a programming language common in the community, such as C, C++ or Python.
3. come with all Makefiles, or equivalent meta-level configuration files for tools such as automake or scons, or both.
4. be demonstrated to work the reference platform of Linux Fedora Core 5 or higher, though it will be developed on a variety of platforms and is therefore likely to be very portable.
5. be thoroughly documented in-line with Doxygen.
6. be delivered with a complete version control history database, in CVS or SVN.
7. depend only on well-known Free or Open Source Software components, compilers, etc.

These requirements are standard practice for all Autonomy Lab projects and the Player Project.

2.2.2 Requirements for simulation model deliverables

The simulation platform will be the well-known, Open Source, Stage multi-robot simulator, of which Vaughan is the lead author and maintainer. Stage runs on the reference Linux platform, and many others. All models and simulation set-ups, environments, robot configurations, etc, used for experiments will be provided as text files and will work with the mainstream release of Player and Stage.

As co-founder and co-designer of Player, Vaughan has more than 7 years of experience developing and experimenting using Player and Stage, and all Autonomy Lab students are trained in using and extending P/S for their experiments.

Chapter 3

Modeling Autonomous Sustain and Resupply

3.1 Modeling the problem

Our first step is to propose a formal model of the problem which captures the essential nature of the informal description given in the contract, compactly but without losing generality. We also aimed to have a principled metric for evaluating our solutions. This process was considerably more difficult time consuming than we anticipated, but increased our understanding of the problem considerably.

Intuitively, the model states that we have a set of resource types and a population of mobile agents, each of which has an individual resource carrying capacity, and a function that describes how the resource load changes over time. The function depends on the agent's position, speed, and the current resource load. If an agent has one or more resources that increase over time, the agent is a producer. If an agent has one or more resources that decrease over time, the agent is a consumer. The consumption of some resources by some agents generates *value*, and the system goal is to maximmize the total value generated over time. Other resources (e.g. energy in the form of battery charge) are consumed but do not generate value - they are the system overhead. Agents can meet in pairs to trade resources, and each pair has a maximum rate at which each resource type can be exchanged. The system goal is to generate motion plans for each robot such that the resources that exist in the system at startup, or are subsequently generated, are consumed to create the largest possible value.

Using this simple but powerful scheme, we can represent a wide variety of producer-transporter-consumer scenarios simply by selecting the appropriate resources, resource-varying functions and other parameters. The following is the formal description of the model.

3.1.1 Formal model

Resources There are m resource types indexed by j.

Agents There are n heterogeneous agents which include all consumers, producers and transporters. Agents are indexed by i. They could be seen as autonomous point-agents with spatial position of agent i denoted by $r_i(t) \in \mathbb{R}^2$ and dynamics $\dot{r}_i(t) = v_i(t), \|v_i(t)\| \leq V_i$. Stationary agents (fixed sites) have $V_i = 0$. Agent i has a stock of $\vec{g}_i(t) = (g_{ij}(t))$ amount of resource j at time t. At all time the load of agent should satisfy load validity predicate G_i, i.e. $\forall t G_i(\vec{g}_i(t))$. A pair of agents can exchange resources and the maximum exchange speed for agents i, k and resource j during any exchange is $X_{ikj} = X_{kij}$.

The stock of resources evolves as resources are produced and consumed by agents and the speed $f_{ij}(\dot{r}_i(t), r_i(t), \vec{g}_i(t))$ of production or consumption depends on the current location, speed and resource stock of an agent.

Exchanges Every agent has an associated set E_i of exchanges with other agents. An exchange

$$e = (t_s, t_e, p, \vec{\Delta g}), e \in E_i,$$

where t_s is the start time, t_e is the end time, p is the index of an exchange partner agent and $\vec{\Delta g}$ is the change vector should satisfy the following conditions:

$$\forall t \in \tau(e) : r_i(t) = r_p(t) = r_i(t_s), \text{where } \tau(e) = [t_s, t_e). \text{ (partners are collocated and immobile)} \quad (3.1)$$

$$t_e \geq t_s + \sum_{j=1}^{m} (|\Delta g_j| X_{ipj}), \text{(feasibility of exchange duration)} \quad (3.2)$$

$$\forall j : g_{ij}(t_e) = g_{ij}(t_s) + \Delta g_j + \int_{t_s}^{t_e} f_{ij}(0, r_i(t), \vec{g}_i(t))dt, \text{(resource accounting)} \quad (3.3)$$

$$G_i(\vec{g}_i(t_e)), \text{(validity of load after exchange)} \quad (3.4)$$

$$\exists e' \in E_p : e' = (t_s, t_e, i, -\vec{\Delta g}), \text{(reciprocity condition)} \quad (3.5)$$

All exchanges should be temporally disjoint: $\cap_{x \in E_i} \tau(x) = \emptyset$.

Resource changes outside exchanges If an agent is not participating in exchange, his resource stock changes as:

$$\forall t \notin \cup_{x \in E_i} \tau(x) : \dot{g}_{ij}(t) = f_{ij}(\dot{r}_i(t), r_i(t), \vec{g}_i(t)), \quad (3.6)$$

3.2 Performance metrics

Gross system value Assuming every unit of resource j produced or consumed by agent i has a current value of P_{ij} the ad-hoc gross system value of the system working from t_0 till t and calculated at t_0 is

$$GSV_{t_0}(t_0, T) = \sum_{i=1}^{n} \sum_{j=1}^{m} P_{ij} \int_{t_0}^{T} f_{ij}(\dot{r}_i(t), r_i(t), \vec{g}_i(t)) e^{-\beta t} dt, \quad (3.7)$$

where β is a discount factor. Discounting is used to model the descreasing value of work done over time.

We can also calculate the post-hoc value at T:

$$GSV_T(t_0, T) = \sum_{i=1}^{n} \sum_{j=1}^{m} P_{ij} \int_{t_0}^{T} f_{ij}(\dot{r}_i(t), r_i(t), \vec{g}_i(t)) e^{\beta(T-t)} dt, \quad (3.8)$$

We can normalize this performance by dividing it by the theoretically possible maximum. The maximum can be achieved if resources in the system can be momentarily exchanged between the agents to maximize the resulting discounted value as if the system has an ideal instantaneous optimal resource transportation system:

$$GSV_{t_0}^{\max}(t_0,T) = \max_{u_i(t),\alpha(t)} \sum_{i=1}^{n}\sum_{j=1}^{m} P_{ij} \int_{t_0}^{T} f_{ij}(\dot{r}_i(t), r_i(t), \vec{m}_i(t))e^{-\beta t}dt, \text{s.t.} \tag{3.9}$$

$$m_{ij}(t) = \alpha_{ij}(t)\sum_{i=1}^{n} g_{ij}(t), \tag{3.10}$$

$$\forall t : \sum_{i=1}^{n} \alpha_{ij}(t) = 1 \tag{3.11}$$

where $\alpha_{ij}(t)$ is the share of total resource j available in the system available for agent i at moment t and $m_{ij}(t)$ gives the corresponding resource amount. $GSV_T^{\max}(t_0,T)$ can be defined similarly.

System value index We define normalized performance indices as

$$PI_{t_0}(t_0,T) = \frac{GSV_{t_0}(t_0,T)}{GSV_{t_0}^{\max}(t_0,T)} \tag{3.12}$$

$PI_T(t_0,T)$ can be defined similarly.

After extensive consideration of the model and metrics, we propose this metric rather than the simpler measures of throughput and efficiency that appeared in our proposal, as it better captures the true "goodness" of the system, expressed in terms of the total time-dependent value of the resources used by the consumers.

Chapter 4

Related Work

This chapter presents a high-level review of the state of the art in aspects of the ASR problem. Each of the later chapters of the report contains its own self-contained review of specific aspects of the problem studies during this period.

4.1 Planning

Most candidate solutions to resource transportation and delivery problems rely on planning as a core component. The problem is expressed in terms of finding in advance a sequence of actions that can be executed to satisfy a set of orders in the best possible way, i.e. minimizing an *a priori* cost function. Simple resource delivery problems are among the set of the standard toy problems used to illustrate various planning approaches and alogrithms [68].

Single-agent discrete time planning is directly applicable to solving resource delivery problems with a single transporter, discrete resources and a set of static producers and consumers. A good survey of the classical planning system STRIPS sufficient for solving this class of problem is given in [25].

Discrete time representation is not appropriate in many cases, and temporal planning is needed to succesfully solve problems with durative actions and resource utilization. One special case of temporal planning is job shop scheduling which is applicable in cases where the partial order on the sequence of actions is known. A compact but thorough survey of various techniques for job shop scheduling is given in [46]. Some problems could be solved by hybrid planning and scheduling systems such as ISIS described in [28]. High-level action languages capable of operating with time and resources were developed as well, but usually planners require domain specific heuristics to operate with them [47, 30].

Introducing several agents capable of parallel operation gives another dimension to the planning problem. Centralized planning for multiple agents (multibody planning) can be performed by using partial order planning extended to handle action concurrency [13]. The latest version of Planning Domain Description Language , PDDL 2.1 [27] can model metric time and continous resource utilization. Several planners that accept problems described in PDDL 2.1 were created and are freely available [23, 37].

Another aspect of muti-agent planning is the possibility to distribute not only plan execution, but the planning itself between different agents. Different coordination mechanisms exist to ensure the consistency of resulting plan. Weiss [94] provides a comprehensive review of coordination mechanisms and other aspects of multi-agent planning. Wilkins [95] describes a complete high-level system architecture for distributed planning. MAPL framework extends PDDL with qualitative temporal relations (capturing actions with uncertain duration) and allows syncronization on communicative acts [14], however no planner implementations of MAPL problems are currently available.

4.2 Multi-Robot systems

The ASR problem is explicitly stated as a multi-robot system problem. The ultimate goal is to produce robot controllers that run on each of a group of robots, such that the overall system behaviour results in resources being distributed appropriately. In addition to the robot controllers, we may have other programs running on ancillary computers to manage coordinattion and planning. Hence we necessarily have a distributed system of robots, loosely coupled through their interactions in the world, but also explicitly coupled through communication.

There is an extensive literature and research community in multi-robot systems. The canonical survey is [86], though this pre-dates a lot of interesting work. Possibly the earliest study of interacting electronic robots was by Walter, who describes the complex interactions of his robots Elsie and Elemer, controlled by "synthetic brains": analog valve circuits modeled on neurons [92]. Multi-robot systems are of interest to researchers for several reasons, each of which is relevant to ASR:

1. producing robust systems due to redundancy - unlike a single-robot system, multi-robot systems can potentially survive the failure of one or more robots.

2. scaling up - several robots can do more work than one, though there are limits to scalability in many scenarios due to interference.

3. heterogeneous systems - in some scenarios it may be easier or cheaper to have various specialized robots instead of one do-anything robot.

4. emergent behaviour - the coupled parallel activity of many simple agents can have surprisingly powerful; the complex structures assembled by ants and bees with relatively small nervous systems and without central control are the classic examples [41]. This idea is informally familiar as the "more-than-the-sum-of-its-parts" idea.

5. competition - competition can be a useful optimization mechanism; for example as demonstrated by natural selection. Mult-robot systems are a natural platform for competition, as well as explcit cooperation.

4.2.1 Task Allocation

A central part of the ASR problem is assigning transport tasks to individual robots. This is an instance of the formal multi-robot task allocation (MRTA) problem, which has been the subject of many articles, but as Gerkey argues [33] typically using ad-hoc approaches. Gerkey provides a useful taxonomy of MRTA problems, obtained from combining these basic properties:

1. **single-task robots (ST) vs. multi-task robots (MT):** ST means that each robot is capable of exe- cuting as most one task at a time, while MT means that some robots can execute multiple tasks simulta- neously.
2. **single-robot tasks (SR) vs. multi-robot tasks (MR):** SR means that each task requires exactly one robot to achieve it, while MR means that some tasks can require multiple robots.
3. **instantaneous assignment (IA) vs. time-extended assignment (TA):** IA means that the available infor- mation concerning the robots, the tasks, and the en- vironment permits only an instantaneous allocation of tasks to robots, with no planning for future alloca- tions. TA means that more information is available, such as the set of all tasks that will need to be assigned, or a model of how tasks are expected to arrive over time.

Problems are categorized by a 3-tuple containing one of each of these pairs of identifiers:

For example, a problem in which multi-robot tasks must be allocated once to single-task robots is designated ST-MR-IA.

Unfortunately, the general ASR problem can be any combination of these, depending on the exact constraints imposed on the particular system. Therefore the analysis of the differences between these catagories provied by Gerkey is not immediately useful, except to elimate some approaches that he shows to apply only to particular combinations. However, this work is very useful for understanding task allocation, and surveys and categorizes the previous work in the area.

Gerkey also provides a very useful result in his PhD thesis [32], showing that optimal ST-SR-IA allocation can be achieved using the Hungarian Method; well known in other fields but introduced to this domain by Gerkey. This result renders many other approaches obsolete, for example various sub-optimal auction-based methods which have been popular in the literature. Gerkey's thesis is currently the authoritative survey of static task allocation methods, while a later article provides analysis of dynamic allocation [54].

4.2.2 Rendezvous

To perform resource exchanges, two robots must rendezvous, i.e. be co-located in space and time for some non-zero duration.

The problem of path planning for multiple robots to assemble in one location, the *rendezvous problem*, has already received some attention. Most authors consider the setting where robots have incomplete information about locations of each other which makes it difficult to coordinate and agree on a single meeting point [24, 21, 45, 70]. For example, [3] describes and proves a distributed rendezvous algorithm for robots with limited visibility. Other authors concentrate on selecting a meeting point to ensure that some specific properties hold, or to optimize the formation during the convergence. Lanthier et. al. [52] present an algorithm for finding the meeting point which minimizes the maximum individual travel costs to a single meeting point on a weighted terrain. Smith et. al. [77] describe a scheme which makes the convergence process more organized in a certain mathematical sense.

Our group has done some recent work on this problem. In [100, 98] Zebrowski and Vaughan introduced robot-to-robot recharging, whereby a dedicated 'tanker' robot transports energy from a fixed charging station and transfers it to a worker robot by physical docking.

In [99] Zebrowski, Litus and Vaughan examine the problem of multiple robots finding a single unique meeting point that minimizes their weighted travel costs. This turns out to be an interesting classical problem (the Generalized Fermat-Torricelli Problem) with no closed form solution. We introduced two specialized numerical methods to find good-quality approximate solutions, and also a perfectly scalable, local-gradient-descent method that provably achieves the rendezvous in time proportional to the distance between the furthest pair of robots. In that paper we assume perfect shared knowledge about the robot locations and shows an effective algorithm which minimizes total energy spent by the robot team in performing their rendezvous. This problem is different from that solved by [52], who instead minimized the maximum energy cost to any individual robot. Our scheme optimizes *energy efficiency*, which is critical for some applications.

We characterize the rendezvous problem as follows:

Assume n robots are located at positions r_i, $i = 1 \dots n$. When a robot moves, it expends energy proportional to the length of its trajectory. Robots have individual energy costs c_i per unit of traveled distance, thus if robot i moves from a to b, it spends $c_i||a-b||$ units of energy. Now the task is to find a point p^* which minimizes the total energy spent by all robots for meeting at that point:

$$p^* = \arg\min_p \sum_{i=1}^{n} c_i||p - r_i|| \tag{4.1}$$

Problem (6.1) is known very well in optimization, where it belongs to the family of *facility location* problems. Historically, it has a variety of names including the Fermat-Steiner problem, Weber problem, single facility location problem, and the generalized Fermat-Torricelli problem. Though it is not possible to find a closed-form solution for this problem, the properties of its solutions are well known (see [36] for the case $n = 3$ and [51] for the general case). Effective numerical algorithms exist [67]. Interestingly, it is also possible to describe the solution using a mechanical interpretation as a system of idealized strings, pulleys and weights [66].

We will now briefly state the properties of the solution point. First, p^* obviously can not be located outside the convex hull formed by r_i. Second, if points r_i are not colinear (not lying upon a straight line), the goal function in (6.1) is strictly convex, which ensures the uniqueness of p^* if $n \geq 3$. Finally, it is possible to prove the following theorem [51].

Theorem 1. *If r_i are not colinear and for each point r_i*

$$||\sum_{j=1}^{n} c_j \frac{r_j - r_i}{||r_j - r_i||}|| > r_i, i \neq j \tag{4.2}$$

then $p^* \neq r_i$ for any i and $\sum_{i=1}^{n} c_i \frac{p^* - r_i}{||p^* - r_i||} = 0$ (the floating case). If (6.3) does not hold for some r_i then $p^* = r_i$ (the absorbed case).

Intuitively, in the floating case all robots meet at some point which is not the starting point of any robot: all robots move. In the absorbed case, one robot stays still and the others drive to it.

If r_i are colinear, then there may exist more than one point where minimum energy cost is incurred and this method may fail. This is not a significant practical problem for two reasons. First, in most real world situations, the robots are unlikely to be perfectly aligned (with the exception of 1-degree of freedom robots such as trains). Second, even in the colinear case the "local dynamic" method presented below converges to a unique instance of the possible minima.

Multi-robot multi-place rendezvous: Frugal Feeding

Having proposed an heuristic solution to the rendezvous problem, we introduce and tackle the much harder problem in [57] whereby a single tanker or service robot must visit each worker robot, but not necessarily at the same location, while minimizing the travel cost of the whole team: the Frugal Feeding Problem.

The Ordered Frugal Feeding Problem can stated formally as follows:

Definition 1 (Ordered Frugal Feeding Problem). *Given tanker location $p_0 \in \mathbb{R}^d$, worker locations $r_i \in \mathbb{R}^d, i = 1, \ldots, k$ and locomotion cost functions $C_i : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, i = 0, \ldots, k$ find*

$$\min_{p_1, p_2, \ldots, p_k} \sum_{i=1}^{k} \left(C_0(p_{i-1}, p_i) + C_i(r_i, p_i)\right); \tag{4.3}$$

$$C_i(x, y) = w_i ||y - x|| \tag{4.4}$$

Here $C_0(x, y)$ gives the cost of tanker relocation from x to y, $C_i(x, y)$ gives the corresponding cost for worker i, and

Definition (4) could be amended to require the tanker to return to its original location after attending all workers, perhaps to refuel itself, without affecting the following analysis.

We denote solution points as p_i^* The possibility of several robots being attended in one place is permitted, and captured by the possible coincidence of some meeting points.

The Fermat-Torricelli problem (also called the Steiner-Weber problem) asks for the unique point x minimizing the sum of distances to arbitrarily given points $x_1, \ldots, x_n$ in Euclidean d-dimensional space. Elaborating on the conventions in the Fermat-Torricelli problem literature [51], we identify some special cases of solution points as follows.

Definition 2 (Special cases for solution). *If $p_i^* = r_j$ for some j, we will call p_i^** **worker absorbed***. If $p_i^* = p_0$, we call p_i^** **tanker absorbed***. Otherwise, we call p_i^** **floating***.*

If the meeting point for worker i is worker absorbed, worker i should either remain still and wait for the tanker to come to it ($i = j$), or it should move to the location of worker j ($i \neq j$). If a meeting point is tanker absorbed for worker i, the tanker should not move, and the worker should drive to the original tanker location. Floating meeting points do not coincide with any original robot location. Finally, there may exist hybrid solutions which combine absorbed and floating meeting points.

In [57] we show that the problem of finding the most efficient order in which to meet the workers is NP-hard, but for a given ordering we give a local and perfectly scalable method to find energy-efficient robot paths, where the quality of the route is very similar to those found by a global numerical approximation method. The local method is shown to converge in time proportional to the distance between the farthest robots, and to take constant computation time and memory per robot per update cycle. Clearly the Frugal Feeding problem is a sub problem of some possible solutions to ASR, and we hope our FF approach may be usefully adapted to a wider class of ASR sub-problems.

4.2.3 Ant-inspired Resource Foraging

In [91] Vaughan described a simple method for a team of robots to cooperatively explore an environment to find a source and sink of resources, then navigate between them, optimizing the route to avoid spatial interference between robots. The method was inspired by pheromone trail following in ants, and tolerates dynamic obstacles, robot breakdowns, etc.

This work was extended in [89, 88] to remove the earlier requirement of global localization: the modified method copes with unbounded drift in localization estimates. Provided the localization error growth rate is low enough that a robot can complete a single round-trip reliably, then robots can usefully maintain a joint 'trail' data structure that functions as a dynamic, minimal map.

4.2.4 Recharging

The standard approach to robot recharging is to ignore it completely. A tiny fraction of the mobile robot literature considers extended operation through recharging at all. Of those that do have recharging robots, the most common approach to deciding when to charge is to use a simple threshold, whereby reaching some lower bound on stored energy causes the robot to switch to a recharging mode.

Very early robots included theshold-based recharging [92]. A recent version includes [75, 76] which describes a probe-like mechanical docking mechanism, and a simple explore-then-recharge controller that was intended only to demonstrate the reliability of the mechanism (which successfully docked in 97 out of 100 autonomous trials). The probe mechanism is shown in Figure 4.1.

The iRobot Roomba, Create and related commercial robots have autonomous recharging capability. The method used to select recharging is likely to be based on the system described in United States Patent 7332890 (issued 19 February 2008), which is a simple threshold approach.

The Roomba and Create have simple contacts under the front of the robot, and drive into the dock, making contact with rounded spring-loaded electrode. A low voltage is used for safety. In our experience, the dock is quite reliable, and it is certainly very low-cost. We chose the iRobot dock for our robot system, which will be used for this project. Figure 4.2 shows the iRobot charger and a Create robot.

The simplicity of the threshold approach is appealing, but it may not be the optimal strategy. For example our work in [93] showed that, in a realistic surveying task, an adaptive threshold produces a higher overall work rate than a static threshold, and a rate maximizing approach outperforms both by a large margin. The biologically-inspired rate maximizing method performs well, but it some important limitations that are discussed in Chapter 7.

In [93] Wawerla and Vaughan analyze the problem of deciding when and how to recharge a single robot during a long-duration survey task. We introduced a simple heuristic method based on a model of foraging behaviour from the behavioural ecology literature, which is shown empirically to perform within 1 or 2% of the optimal solution discovered by brute-force search. The heuristic is fast and perfectly scalable, using a greedy strategy of maximizing the *rate* at which energy is gathered over time. Intuitively, maximizing the rate at which energy is input into the robot means we can maximize the rate at which energy is output in useful work.

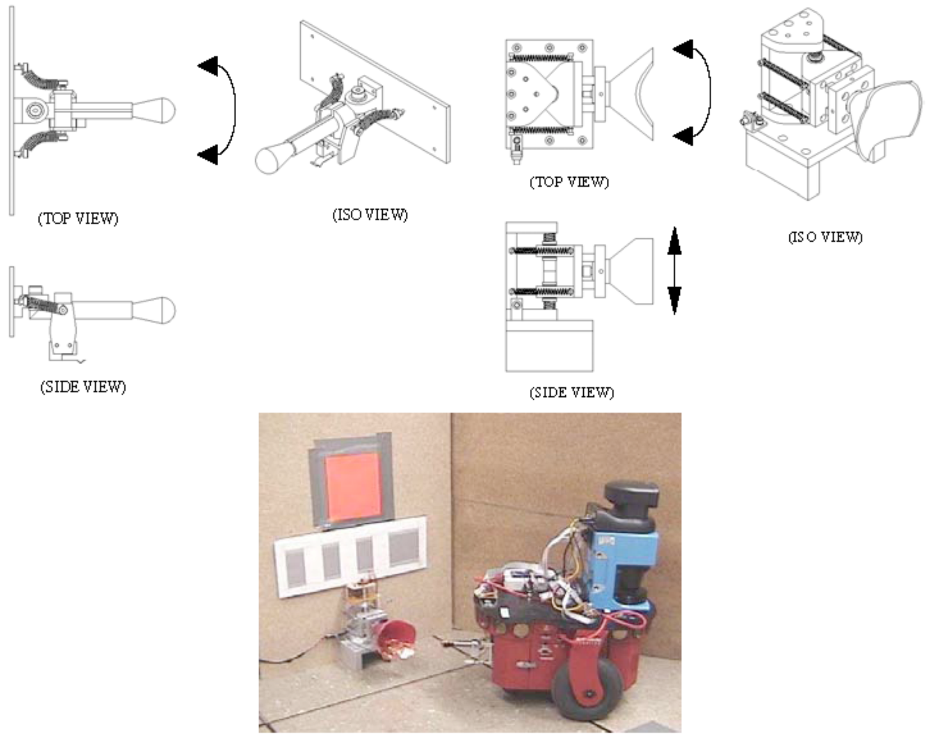


Figure 4.1: Recharging dock mechanism from [76], showing schematics of male and female components, and implementation fitted to a Pioneer robot. Images copyright IEEE.

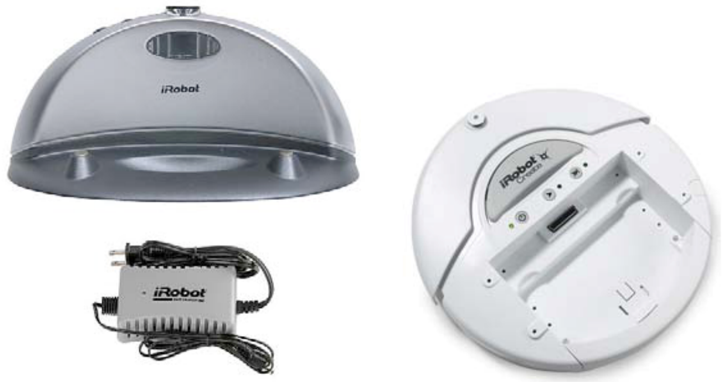


Figure 4.2: Recharging dock for the iRobot Create robot. The Create has simple contacts under the front of the robot, and drives into the dock, making contact with rounded spring-loaded electrode. Images copyright iRobot Corp, taken from their web site.

4.2.5 Interference Reduction

A key problem for multi-robot systems working in constrained environments, or when robot spatial density is high, is spatial interference that degrades performance. The standard approach is planning: robot trajectories are selected such that robots do not interfere with each other. Alternative approaches have been suggested that are more scalable, mainly based on the idea of territoriality, so that robots operate in distinct work zones, and avoid each other. This is often considered in a context of foraging for resources, the most representative study being [26].

In a series of papers and theses including [90, 16, 103, 102] we have explored the used of local robot-to-robot signaling strategies to break symmetry and resolve conflicts over resources. We have shown several refinements of an "aggressive display" technique whereby robots indicate their "aggressiveness" to each other, and the more aggressive robot gains access to the right of way in a narrow corridor. While setting aggression at random is a good general technique for breaking symmetry, we have invented several methods that favour the globally "best" robot with high probability, while using only local information. Such rational aggression is shown to significantly increase the throughput of the population of robots performing a resupply task in a confined space like an office building or narrow trail. The aggressive interaction method is applicable to any conflict over resources, where cheap local symmetry-breaking is required.

Chapter 5

Classical solution

5.1 Abstract description

It is natural to functionally decompose the problem into two interacting abstraction levels. At the high level we abstract from the details of between-exchange travels. At this level we view the problem as if agent moving between two exchange locations is a deterministic action of known duration and required resource expenditure. At the high level the problem becomes a problem of multi-agent temporal planning with continuous time. A plan for agent i is a sequence of exchanges $(e_{i1}, e_{i2}, \ldots, e_{il_i})$. The sequence can be empty, thus leaving some of the agents idle. During the plan execution conditions might change (new agents appear or become damaged, agents are removed from the system, travel time estimate turns out to be wrong, etc.). Therefore, the system should perform execution monitoring and replanning thus achieving continuous planning during the system life. Hence, solving the high level problem gives each agent the high-level task of moving from current location to the exchange location and performing an exchange. This task presents the low abstraction level of the problem. It consists of point-to-point path planning, safe navigation with dynamic obstacle avoidance, global localization and speed control and potentially some automatic exchange support. The general architecture of the system is shown at Fig 5.1.

5.2 Implementation

In this section we informally outline possible implementations of different modules in a classical solution. We attempt to create a system minimizing the set of modules that should be custom made. Using standard off-the-shelf solutions will make the system support and upgrade cheaper and more available.

Planning task generator The planning task generator performs two activities. First, based on the description of the current agents state and their properties as well as the resource values supplied exogenously it devises the set of delivery orders. These orders are the changes in resource loads of a subset of agents that, if performed, will improve the performance metric of the system. Particular mechanism that decide which set of orders should be served depends on the class of possible resource dynamics functions f_{ij} and the representation of these functions. These system design decisions influence the nature of

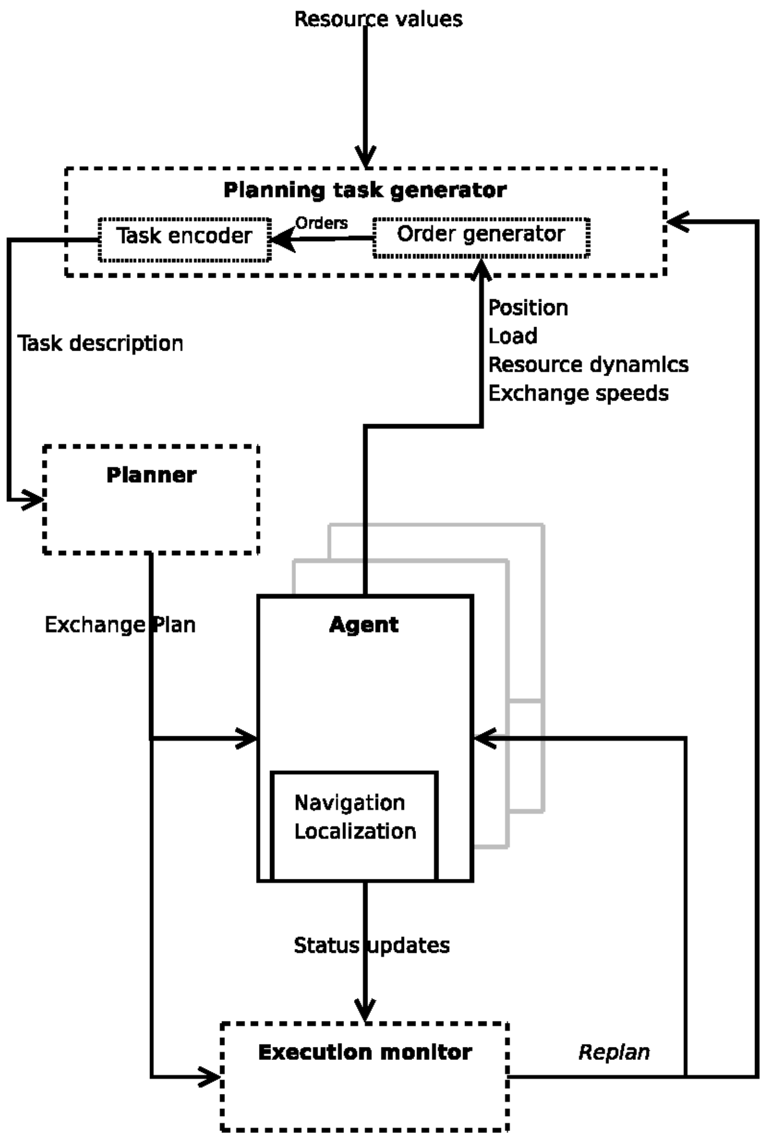


Figure 5.1: System architecture. Resource values P_{ij} are provided exogenously. Planning task generator submits a planning task to the planner, which outputs a plan E_i for every agent. Execution monitor receives status updates from the agents which include information about performed exchanges, new agent registration and malfunction notices. Monitor makes the decisions about needed replanning. If replanning is needed, agents report their current positions $r_i(t)$, loads $\vec{g}_i(t)$, resource dynamics functions f_{ij} and exchange speeds X_{ikj} to planning task generator and the cycle repeats.

the optimization task to be solved and algorithms that will be used for this optimization. Optimization is performed by the order generator subsystem. The set of resulting orders is passed to the task encoding subsystem. There is an override mechanism that allows to supply the set of orders to be served directly to the task encoding subsystem. The task encoding subsystem is responsible for encoding the set of orders as a planning task in some formal planning description language capable of describing multi-agent temporal domains. PDDL 2.1 described in [27] is a good candidate. Most of the planning languages separate domain description and problem description, thus only problem description should be regenerated every time a new set of orders arrive. Problem description has point-to-point travels and exchanges as possible actions and serving all orders as a goal. The generator calculates the travel duration to static agents or approximates the rendezvous time for pairs of mobile agents. This information is used to provide action durations to the planning problem description. Depending on the planner resources may need to be discretized to make a set of possible exchanges finite.

Planner The planner receives a problem description in a formal planning language and finds a good plan that satisfies described goal. We intend to use the freely available centralized planner described in [47] that solves problems defined in PDDL 2.1 language. Later we plan to consider possible distributed planning solutions. Once the planner finds a good plan, exchange sequences are communicated to involved agents.

Execution monitor The execution monitor receives the plan from the planner and status update messages from agents. The monitor compares the actual start and end time of the exchanges with the planned time. If a deviation is detected, the monitor estimates the impact on the performance. If the deviation is significant to cause an impact which exceeds a certain threshold, a replanning command is sent to all agents and the planner.

Localization We assume high-quality global localization is available. This is reasonable in the state of the art, and can provided outdoors with GPS and indoors with Monte-Carlo localization methods, as referenced in the literature review.

Navigation Point to point navigation can be provided by various standard means. For basic experiments we can use the wavefront navigation planner provided with the Player distribution, though if we need to examine terrain with various motion costs we will need to add additional functionality to Player's implemntation.

Obstacle avoidance will use Player's Vector Field Histogram [85] or Nearness Diagram [61] modules. We do not intend to focus on this much-studied area.

Chapter 6

Distributed Gradient Optimization with Embodied Approximation

Authors: Yaroslav Litus and Richard Vaughan

6.1 Abstract

We present an informal description of a general approach for developing decentralized distributed gradient descent optimization algorithms for teams of embodied agents that need to rearrange their configuration over space and/or time, into some optimal and initially unknown configuration. Our approach relies on using embodiment and spatial embeddedness as a surrogate for computational resources, permitting the reduction or elimination of communication or shared memory for conventional parallel computation. Intermediate stages of the gradient descent process are manifested by the locations of the robots, instead of being represented symbolically. Each point in the space-time evolution of the system can be considered an approximation of the solution, which is refined by the agents' motion in response to sensor measurements. For each agent, motion is approximately in the direction of the local antigradient of the global cost function. We illustrate this approach by giving solutions to two non-trivial realistic optimization tasks from the robotics domain.

We suggest that embodied approximations can be used by living distributed systems to find affordable solutions to the optimization tasks they face.

6.2 Introduction

We know by now that The World Is It's Own Best Model ([15]). Brooks' recommendation to avoid internalizing the world - a costly and error-prone process - but rather to sense and react to it directly whenever possible, has become part of the robotics canon. This paper makes explicit an inversion of this approach, in which where the world is used to externalize a computational process.

In particular, we present an informal description of a general approach of developing decentralized distributed gradient descent optimization algorithms for teams of embodied agents that need to rearrange their configuration over space and time into some optimal and initially unknown configuration. This class of problems includes many classical mobile agent problems such as formation control, facility location, rendezvous, navigation and interference reduction. Given the intractability of finding optimal solutions to these large, high-dimensional joint state-space planning problems, gradient descent methods are commonly used to find approximate solutions.

The proposed approach is to use embodiment and spatial embeddedness as a surrogate for computational resources, permitting the reduction or elimination of communication or shared memory for conventional parallel computation. Intermediate stages of the gradient descent process are manifested by the locations of the robots, instead of being represented symbolically. Each point in the space-time evolution of the system can be considered an approximation of the solution, which is refined by the agents' motion in response to sensor measurements. For each agent, motion is approximately in the direction of the local antigradient of the global cost function.

Gradient-based formation control and navigation have been widely used in robotics [101, 82]. The idea of embodied computation was recently presented in [38], however contrary to the authors' claim we believe that globally defined algorithm can be implemented by embodied computation. Finally, [58] have shown biologically plausible navigation and tracking algorithms which involve using other agents location to calculate local gradient. To our knowledge, this paper is the first to explicitly present embodied approximation as a method to implement parallel embodied gradient optimization algorithms.

We illustrate this approach by giving solutions to two non-trivial realistic optimization tasks from the robotics domain. This work is in the context of our interest in large-scale distributed systems such as animal colonies and multi-robot systems, which work together to solve complex tasks. We are interested in identifying mechanisms that exploit the characteristics of the embodied multi-agent domain to solve complex computational problems. We are particularly interested in solving practical resource allocation problems in multi-robot systems, with *energy autonomy* as the key motivating problem. This is the motivation for our choice of example problems: two different versions of energy-efficient robot-robot rendezvous, useful for for recharging or refueling, or as a component of various other tasks. While looking for ways to find meeting places which minimize the traveling costs for groups of robots we developed fully decentralized heuristic methods which also require no range information to converge at good approximation to the optimal solution. These heuristics afford a very simple implementation.

The key insight that underlies our approach is that the spatial configurations of the agents themselves could be considered as an approximate solution to the entire problem. An individual agent can move itself, thus refining the component of current solution approximation. No representation of the problem, or the current solution, needs to be held by any robot: they manifest the solution by their physical configuration. This is an example of what Payton has called "world-embedded computation" [64] and exploits the property of "strong embodiment" identified by [15], extended to a multi-agent system.

6.3 Distributed gradient optimization with embodied approximation

Assume a system of n spatially embedded agents is described by a spatial configuration $s \in H$ where $H = H_1 \times H_2 \times \ldots \times H_n$ and H_i is a vector space of possible configurations of agent i. We will denote the i-th component of s as s_i. Every agent has control over the first-order dynamics of its own configuration which allows it to continuously change its configuration possibly with some restrictions on the speed of this change. We also assume that every agent i can sense the spatial configuration of a set of other agents ψ_i. This set can change as the system evolves. Given some cost function $J : H \rightarrow R$ we want to rearrange the system into an optimal configuration $s^* = \arg\min_s J(s)$. Note that the cost function may be parametrized by initial agents configuration s_0.

If the function J has certain mathematical properties (e.g. is continuously differentiable) then a good approximation to optimal configuration can be found in a centralized way by using one of many gradient optimization algorithms. It will start with some initial approximation and iteratively change it in constant or decreasing steps in the direction of (approximated) antigradient. Details of the particular algorithms are irrelevant for present discussion. This centralized solution will need a central processor with amount of memory of the order of the size of the state vector s as well as means to communicate s^* to all agents once an acceptable approximation is found. Once all agents know their component of s^* they can proceed directly towards it. If J is parametrized by initial configuration of agents S_0 then the means of communicating or sensing this configuration by the central processor are required.

Since the system of n agents in question has n processors that can work in parallel it is natural to try to use this resource to get a parallel solution. Distributed implementation of iterative algorithms are well-studied [10, 11, 50]. It is known that if processors synchronously compute and communicate their partial results to other processors then resulting distributed procedure is equivalent to a single-processor implementation thus preserving its convergence properties. Even more, under certain realistic conditions (like small iteration steps and bounded communication delays) the synchrony assumption can be relaxed and processors can be allowed to compute at different speeds and communicate sporadically while still giving the same convergence properties as original serial algorithm [84]. Therefore, we can assign every agent i the task of iterative optimization of its own component s_i of the global state vector s by changing s_i in the direction of the corresponding component of the antigradient vector ∇J_i and periodically communicate current value of s_i to other agents as well as receive updates of the current approximations of $s_j, j \neq i$ from other agents. Note, that the amount of information from other processors needed to compute ∇J_i depends on the particular problem. If ∇J_i depends only on s_i then no additional information is necessary and s_i can be computed independently of other processors. If ∇J_i depends on some other components of global state s then the current values of these components should be received from the processors which compute them. We denote the set of agents that compute components necessary to calculate ∇J_i by ξ_i. If the assumptions of [84] are met, then eventually every agent will know the approximation s_i^* and will be able to proceed towards it.

However, a physical multi-agent system is much more than simply n parallel processors. Physical embodiment implies spatial embeddedness, and for some problems these remarkable dual properties allow us to drastically reduce or totally eliminate the need to communicate intermediate results of calculations, or indeed any *decription* of the problems or solutions, substituting instead direct physical observations. In addition, instead of waiting for an iterated algorithm to converge agents can perform reconfiguration *during* the optimization thus giving the resulting algorithm an attractive anytime property. The high-level description of the approach is given in Algorithm 1.

Algorithm 1 Gradient optimization with embodied approximation

```
1: for all agents $i$ do
2:   sense current configurations $s_j^{\psi}$ for $j \in \psi_i \cap \xi_i$
3:   update using communication current configurations $s_j^{\psi}$ for $j \in (\{1, 2, \ldots, i-1, i+1, \ldots, n\} \backslash \psi_i) \cap \xi_i$
4:   $D_i \leftarrow \nabla J_i(s_i, s_{j|j \in \xi_i})$
5:   if $\nabla J_i$ exists and component can be improved then
6:     move in direction $-D_i$
7:   else
8:     stay still
9:   end if
10:  communicate new configuration to other agents
11: end for
```

The key idea is to use agent configurations directly as approximations to corresponding components of global goal configuration. Thus all agents move in the (approximated) antigradient direction which is calculated at Step 4 using their current configuration as the current approximation to the solution. The communication at Step 3 is necessary only if not all of the information from the relevant agents belonging set ξ_i can be acquired by direct sensing at Step 2. In the best case no communication is required as all necessary information is available via sensing, so $\psi_i \supseteq \xi_i$. If some components nevertheless have to be acquired by communication, this could be done in a asynchronous sporadic manner as was argued above.

This approach makes agents themselves serve as a shared "memory" for the parallel computation they perform where the information about current approximation is embodied in physical configuration of the agents. Once the values of necessary components are available, gradient direction ∇J_i or an approximation to it can be calculated and agent can move in antigradient direction thus improving approximation of component s_i and global state s. In a certain sense, agent relocation becomes a part of computation process creating a parallel computer comprised by agent processors and physical bodies of agents. The algorithm continues until convergence is reached which is detected in a way specific for particular problem and algorithm employed.

6.4 Applications

We illustrate the idea of optimization with embodied approximation by two problems of energy-efficient multi-robot coordination which we studied in our previous work. The first problem is to assemble a team of robots at an initially unknown point which minimizes the total energy spent on relocation [99]. The second, more difficult problem is an instance of the multi-facility location problem which asks to plan a route for a team of robots to rendezvous a single dedicated service robot, possibly each at a *different* point [56, 57].

6.4.1 Energy-efficient single-point rendezvous

Assume n robots are located at positions r_i, $i = 1 \ldots n$. When a robot moves, it expends energy proportional to the length of its trajectory. Robots have individual energy costs c_i per unit of traveled distance, thus if robot i moves from a to b, it spends $c_i||a - b||$ units of energy. Now the task is to find a point p^* which minimizes the total energy spent by all robots for meeting at that point.

Definition 3 (Energy-efficient rendezvous problem). *Given robot locations $r_i \in R^d, i = 1, \ldots, n$ find rendezvous point*

$$p^* = arg\min_p J(p) = \sum_{i=1}^{n} c_i||p - r_i|| \tag{6.1}$$

Historically, this problem has a variety of names including the Fermat-Steiner problem, Weber problem, single facility location problem, and the generalized Fermat-Torricelli problem. Though it is not possible to find a closed-form solution for this problem, the properties of its solutions are well known (see [36] for the case $n = 3$ and [51] for the general case). Effective numerical algorithms exist [67]. Interestingly, it is also possible to describe the solution using a mechanical interpretation as a system of idealized strings, pulleys and weights [66].

The antigradient of the cost function is

$$-\nabla J(x) = \sum_{i=1}^{n} u(x, r_i), u(a, b) = \frac{b - a}{||a - b||} \tag{6.2}$$

That is, the antigradient at some point is simply a sum of unit vectors pointing in the directions of original robot locations. Note that antigradient is not defined at the locations of the robots themselves while one of these locations can be a solution. If points r_i are not collinear (not lying upon a straight line), the goal function in (6.1) is strictly convex, which ensures the uniqueness of p^* if $n \geq 3$. In this case solution is characterized by the following theorem [51].

Theorem 2. *If r_i are not collinear and for each point r_i*

$$||\sum_{j=1}^{n} c_j \frac{r_j - r_i}{||r_j - r_i||}|| > r_i, i \neq j \tag{6.3}$$

then $p^* \neq r_i$ for any i and $\nabla J(p^*) = 0$ (the floating case). If (6.3) does not hold for some r_i then $p^* = r_i$ (the absorbed case).

A simple embodied approximation method gives a useful solution to this problem. Let all robots compute the solution point approximation simultaneously by checking the absorbed case condition in Theorem 2 and moving along antigradient direction if the condition is not met. This produces the distributed Algorithm 2 which runs until robots converge to a single point. Note that since robots are embodied they can not occupy the same point, thus we consider that two robots met whenever the distance between them is closer than some meeting range ϵ.

In order to calculate the antigradient at its current location, each robot needs to know the direction towards the original robot locations r_i. This could be achieved either by means

Algorithm 2 Static algorithm for energy-efficient rendezvous

```
1: for all robots $i$ do
2:     Update current location of self, $x_i$
3:     $D_i \leftarrow \sum_{i|x \neq r_i} c_i u(x, r_i), u(a,b) = \frac{a-b}{||a-b||}$
4:     if $x = r_i$ for some $i$ then
5:         $c \leftarrow c_i$
6:     else
7:         $c \leftarrow 0$
8:     end if
9:     if $||D_i|| < c$ then
10:        stop
11:    else
12:        move in direction $D_i$
13:    end if
14: end for
```

of global localization and memorizing r_i or by setting static beacons at points r_i (hence the name of the algorithm). Both localization and static beacons are expensive solutions, but embodied approximation can be used once again to get rid of the necessity to calculate directions to r_i.

Once some of the robots move from their original locations r_i a new instance of the rendezvous problem is created. In this new instance all robots are located at the new "original" locations r_i thus sensing the instantaneous directions to the robots is enough to calculate the antigradient value. Hence memorizing r_i or using static beacons is replaced by using the robots as dynamic beacons resulting in the distributed Algorithm 3.

Algorithm 3 Dynamic algorithm for energy-efficient rendezvous

```
1: for all robots $j$ do
2:     $A_j \leftarrow \{i| ||r_i - r_j|| \leq \epsilon\}$, meaning the set of robots which are closer to $r_j$ than meeting
       threshold $\epsilon$. Note that $j \in A$, thus $A$ has at least one element.
3:     $D_j \leftarrow \sum_{i \notin A} c_i \vec{d}(r_j, r_i)$.
4:     $c \leftarrow \sum_{i \in A} c_i$
5:     if $||D_j|| < c$ then
6:         stop
7:     else
8:         move in direction $D_i$
9:     end if
10: end for
```

A set of experiments was conducted to compare the performance of these algorithms with the performance of a centralized optimization algorithm (see [99] for details). Some typical results are shown in Table 6.1. The centralized static algorithm calculates the optimal meeting point p^* exactly once and commands all robots to proceed directly to that point. The centralized dynamic algorithm periodically recomputes the meeting point incorporating new positions of the robots while they move towards their rendezvous. It could be seen that distributed methods achieve the quality of solution comparable to centralized methods. The difference is explained by the fact that the trajectory along the antigradient is not

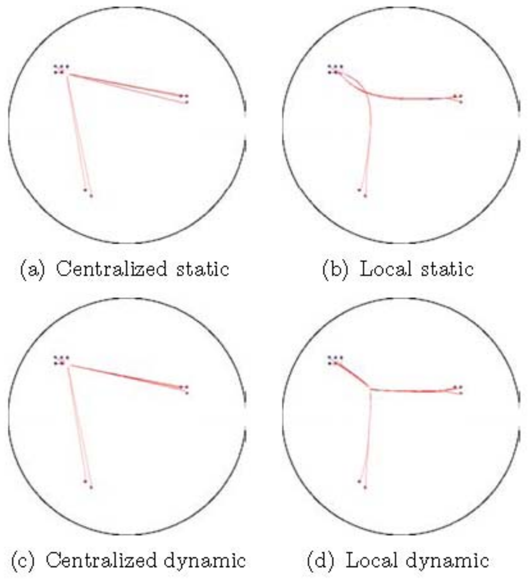

(a) Centralized static (b) Local static (c) Centralized dynamic (d) Local dynamic

Figure 6.1: Typical paths taken to single point rendezvous (Map 2)

Table 6.1: Mean Total Energy Used For Rendezvous. All Standard Deviations < 5%

	Dynamic		Static	
Map	Centralized	Distributed	Centralized	Distributed
1	149.99	151.56	149.92	152.79
2	105.21	115.21	105.25	119.00
3	77.56	80.49	77.28	81.48
4	477.89	503.18	476.97	530.11

straight and thus longer then the direct path to the optimal meeting locations (see Fig. 6.1). However, in applications that require scalability distributed algorithms with low demands on sensing and communication will be preferable even if they produce slightly worse results than centralized solutions.

6.4.2 Ordered Frugal Feeding Problem

Consider a team of worker robots that can recharge by docking with a dedicated refueling (or equivalently, recharging) robot called a *tanker*, as described in [100]. The tanker robot could remain at a fixed location, acting as a conventional charging station, or it could move to rendezvous with worker robots. Simultaneously, worker robots can wait for the tanker to come to them, or they can move to meet the tanker.

By analogy to a mother animal attending her offspring we call this problem the "Frugal Feeding Problem". If we impose a total order in which worker robots must be met and charged, (perhaps based on urgency or some other priority scheme) we obtain the "Ordered Frugal Feeding Problem" considered here. As in a single point rendezvous problem we model locomotion costs as the weighted Euclidean distance between the origin and destination.

The Ordered Frugal Feeding Problem can be stated formally as follows:

Definition 4 (Ordered Frugal Feeding Problem). *Given tanker location $p_0 \in \mathbb{R}^d$, worker locations $r_i \in \mathbb{R}^d, i = 1, \ldots, k$ find*

$$\min_{p_1, p_2, \ldots, p_k} J(p_1, p_2, \ldots, p_k) =$$
$$\sum_{i=1}^{k} \left(w_0 ||p_i - p_{i-1}|| + w_i ||r_i - p_i|| \right); \tag{6.4}$$

Here $w_0||p_i - p_{i-1}||$ gives the cost of tanker relocation between points p_{i-1} and p_i, $w_i||r_i - p_i||$ gives the cost of moving worker i from its original position to meeting point p_i.

Definition (4) could be amended to require the tanker to return to its original location after attending all workers, perhaps to refuel itself, without affecting presented results.

We denote solution points as p_i^* The possibility of several robots being attended in one place is permitted, and captured by the possible coincidence of some meeting points. We define a *meeting* as the event when robots come within distance s of each other.

Unlike the single-point rendezvous case this problem involves several relocations of the system (tanker needs to go between all rendezvous points in turn). However, the embodied approximation approach is applicable and results in a distributed algorithm which produces good approximations to the optimal solutions.

We start with considering the antigradient of the cost function J. Using Eq (6.2) we can express the i-th component of antigradient as

$$-\nabla J_i(p_i) = w_0 u(p_i, p_{i-1}) + w_0 u(p_i, p_{i+1}) + w_i u(p_i, r_i), \tag{6.5}$$

for $i = 1..k-1$. The antigradient component for the last robot k obviously includes one less term. Now let every worker robot i represent current approximation to the solution point p_i^*. Let the tanker also represent the current approximation of the meeting point for the first robot to be charged. That is, point p_1 is simultaneously represented by tanker and worker 1. The system will perform parallel gradient descent with all robots moving along the corresponding approximation to the antigradient component given by Eq (6.5) calculated at their own position. In other words, robots 0 (the tanker) and 1 (the next robot to be charged) move along the approximated antigradient of the part of cost function $g(p_1) = w_0||p_0 - p_1|| + w_0||p_1 - p_2|| + w_1||p_1 - r_1||$ using the current position of robot 2 as an approximation to the unknown solution point p_2. The rest of the robots $j, j = 2, ..., k$ move along the approximated antigradient of the cost functions $f_j(p_j) = w_0||p_j - p_{j-1}|| + w_0||p_j - p_{j+1}|| + w_j||r_j - p_j||$ using r_{j-1}, r_{j+1} as approximations for the unknown solution points p_{j-1}, p_{j+1}. Algorithm 4 describes the procedure formally. Convergence if this algorithm is analyzed in [56].

Parallel computation of movement directions in steps 11-14 and simultaneous movement of all robots in steps 17-20 provide the scalability of this algorithm. Importantly, the per-robot, per-timestep cost is constant, so the method works for arbitrary population sizes. Unlike the single-point algorithms described in previous section, here every robot needs to know the direction to only two other robots to calculate its movement direction.

More specifically, this algorithm requires every worker to know its own weight and the tanker weight, whether or not it is the head of the current charging queue, and the direction towards the next worker in the queue and the previous worker (or tanker if the robot is at

Algorithm 4 Distributed Algorithm for Ordered Frugal Feeding Problem

```
1: $i \leftarrow 1$
2: define $\vec{d}(x, y) = (y - x)/||x - y||$
3: let $r_0$ be the current tanker position, $r_i$ be the position of worker $i$
4: while $i \leq n$ do
5:   if $r_0$ is close to $r_i$ then
6:     tanker charges worker $i$; $i \leftarrow i + 1$
7:   else
8:     if $i = n$ (only one robot in queue) then
9:       the lighter of tanker and worker goes towards the other
10:    else
11:      $r_{n+1} = r_n$
12:      $\vec{D}_0 \leftarrow w_1\vec{d}(r_0, r_1) + w_0\vec{d}(r_0, r_2)$.
13:      $\vec{D}_i \leftarrow w_0\vec{d}(r_i, r_0) + w_0\vec{d}(r_i, r_{i+1})$
14:      for all $i < j \leq n$ do
15:        $\vec{D}_j \leftarrow w_0\vec{d}(r_j, r_{j-1}) + w_0\vec{d}(r_j, r_{j+1})$
16:      end for
17:      for all $j \in \{0, i, i + 1, \ldots, n\}$ do
18:        if $||\vec{D}_j|| < w_j$ then
19:          robot $j$ stops
20:        else
21:          robot $j$ proceeds in the direction $\vec{D}_j$
22:        end if
23:      end for
24:    end if
25:  end if
26: end while
```

the head of the queue). The tanker needs to know its own weight, the weights of the first two robots in the queue and the directions towards them. As a robot is met and charged, it is removed from the queue, and this update is broadcast to all robots.

As in the single point rendezvous case here instead of operating with the model of the world and searching for the complete solution, each robot uses the position of itself, its queue predecessor and successor as the current embodied approximation to the solution points. Every robot tries to improve the global solution quality by moving in the direction which decreases the part of the total cost function that concerns itself and its neighbors. If the robot finds itself located at the minimizing point for the current local configuration of robots, the robot stops. Fig 6.2 shows some trajectories produced by the algorithm.

To evaluate the performance of the algorithm we performed 3000 simulations running distributed algorithm for Ordered Frugal Feeding Problem on randomly generated instances of the problem, with initial robot locations drawn from the same uniform distribution, and 3 different uniform distributions of weights, with 1000 experiments for each weight distribution. In each experiment, 10 randomly weighted worker robots and a tanker were placed at random locations in an square arena with 20m sides. A meeting range of a 0.1m and a movement step length of 0.01m were set. The path traveled by each robot was recorded, and it's length was multiplied by the robot's weight, then summed for the population to give the total energy spent on performing the rendezvous. For comparison

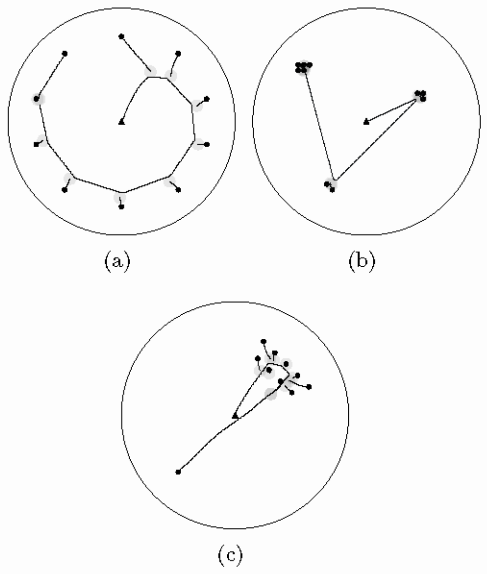


Figure 6.2: Some trajectories produced by distributed algorithm for Ordered Frugal Feeding Problem. Triangle represents tanker, small disks represent workers, large disks show the meeting ranges at the final point of each worker trajectory.

we computed optimal solutions of each instance using discrete search [57] on a regular grid with 40 ticks per dimension covering the embedding hypercube of original robot locations. The regularity of the grid allows us compute a lower bound of the optimal cost based on the result of discrete search by subtracting the maximum possible error due to discretization (no such quality bounds are readily available for numerical approximation methods to the continuous problem, and no closed form solution is known).

We calculated the upper bound of approximation factor for every problem instance by dividing distributed algorithm for Ordered Frugal Feeding Problem results by the lower bound of optimal cost. Table 6.2 reports the statistics of these approximation factor bounds for each distribution.

Table 6.2: Statistics of approximation factor bounds

	$w_i = 1$	$w_i \sim U[1, 3)$	$w_i \sim U[1, 100]$
Mean	1.19	1.22	1.31
Median	1.19	1.20	1.25
St. dev.	0.03	0.08	0.23
Skewness	0.67	1.18	4.27
Kurtosis	0.65	1.65	31.82

The results show increasing variation of the approximation factor bounds with increasing variation in robot weights. Indeed, distributed algorithm for Ordered Frugal Feeding Problem can cause the tanker to move suboptimally in the direction of two light workers and then return to the third heavy worker. Hence, distributed algorithm for Ordered Frugal Feeding Problem does not have a constant approximation factor. However, the average quality of solutions for uniformly distributed problem appears very good. Thus, the method could be used where the guaranteed quality of results obtained in non-scalable centralized manner should be sacrificed in favor of simple decentralized scalable solution with good average performance.

6.5 Conclusion

A general approach for development of decentralized distributed gradient descent optimization algorithms for teams of embodied agents is presented. This approach uses embodiment and spatial embedding as valuable computational resources which allow to reduce or eliminate communication or shared memory requirements for parallel computation. Spatial configuration of agents serves as an embodied representation of the current approximation to global solution which is accessed by means of sensing and refined by agents moving along local antigradient directions. Two non-trivial optimization tasks of energy-efficient path planning for mutli-robot teams were used to illustrate the embodied approximation approach. Resulting distributed algorithm show very good results in experimental evaluation. These algorithms use very simple mechanisms that result in energy efficient group behavior. In both cases, computationally complex optimization tasks are solved using biologically affordable machinery of bearing-only sensing. Thus, it seems promising to look for examples of optimization by means of embodied approximation in groups of animals. Such animal strategies could and should be used to increase the efficiency, and eventually autonomy, of robot teams.

Admittedly the extent to which embodied approximation can substitute communication or shared memory is limited by the properties of sensors and environment. Limited sensor range and occlusions will limit the number of state vector components that could be accessed by sensing. However, some optimization problems by their nature need only local and limited exchange of information. For the rest of the problems embodied approximation may still significantly reduce the need for inter-agent communication and should be considered as one of the available resources.

Future work includes application of embodied approximation approach to other optimization problems emerging in multi-agent teams. A search for biological examples of parallel gradient optimization with embodied approximation is another interesting direction. Finally, accurate mathematical models of spatially embedded multi-agent systems with certain sensing capabilities could be developed and the computation power of such systems can be theoretically studied taking into account embodied approximation as a computational resource.

Chapter 7

Optimal Robot Recharging Strategies for Time-Discounted Labour

Authors: Jens Wawerla and Richard Vaughan

7.1 Abstract

Energy is defined as the potential to perform work: every system that does some work must possess the required energy in advance. An interesting class of systems, including animals and recharging robots, has to actively choose when to obtain energy and when to dissipate energy in work. If working and collecting energy are mutually exclusive, as is common in many animal and robot scenarios, the system faces an essential two-phase action selection problem: (i) how much energy should be accumulated before starting work; (ii) at what remaining energy level should the agent switch back to feeding/recharging? This paper presents an abstract general model of a energy-managing agent that does time-discounted work. Analyzing the model, we find solutions to both questions that optimize the value of the work done. This result is validated empirically by simulated robot experiments that agree closely with the model.

7.2 Introduction

Since the early years of robotics, e.g. [92], roboticists have been challenged by the need to supply robots with energy. For mobile robots that carry their own limited energy store, the key question is "when should the robot refuel/recharge?". A standard approach in the literature and in commercial robots is to set a fixed threshold and refuel when ever the robot's energy supply drops below this threshold ([75], [76]). The simplicity of this approach is appealing, but it may not be the optimal strategy. For example [93] showed that, in a realistic surveying task, an adaptive threshold produces a higher overall work rate than a static threshold, and a rate maximizing approach outperforms both by a large margin. The biologically-inspired rate maximizing method performs well, but it some important limitations discussed below. This paper provides a more sophisticated model for rational recharging robots

Robots, by their name[1] and nature, are supposed to perform some form of labour, e.g. space exploration, entertainment, rescue missions, clean-up, assembly, etc. Most tasks require execution in a timely manner, though some more then others. For example, we may be willing to wait for a day or two for the latest geological observations from Mars, while waiting the same time for the rescue of trapped miners or ordnance disposal might not be acceptable. The standard method to model the descreasing value of work over time, and thereby encourage timely execution, is to discount by some factor β in discrete timesteps the reward given in exchange for investment ([87]), where investment here is energy dissipated in labour. The inverse of discount $(1/\beta)$ is the familiar interest rate, of savings accounts and credit cards.

The laws of physics dictate that energy cannot be transferred instantaneously, or in other words, refuelling takes time and this time cannot be spent working. If a robot spends an hour refuelling, it starts to work one hour later and since the reward is discounted over time, it receives a smaller payment then if it would have started working immediately. But the initial charging period is strictly required, as no work can be done without previously obtaining energy. This conflict between the mutually exclusive tasks of refuelling and working raises two interesting questions:

Q1 How much energy should be accumulated before starting work?

Q2 At what remaining energy level should the agent switch back to obtaining energy?

Most real-world robot systems avoid these questions by maintaining a permanent connection to an energy source, e.g. industrial robotic manipulators wired into the mains power grid, or solar powered robots which are capable of gathering energy while performing some task at the same time. This paper addresses the more interesting class of machine, including animals and mobile robots, that must obtain and store energy prior to working. The Q1,Q2 action selection problem must be solved by every animal and long-lived robot in some way or another. Further we are only considering rational agents. Any introductory text book on decision maring (e.g. [81]) defines an agent to be rational if it always selects that action, i.e. an answer to Q1 and Q2, that returns the highest expected utility. Here we assume that utility is proportional to the reward obtained by working, which is discounted over time.

After considering related work, we analyze the problem in terms of a simplified abstract model which, when parameterized to approximate a particular robot system, predicts the optimal answers to Q1 and Q2. We validate the model by comparing its predictions with data empirically obtained from a simulated robot.

To the best of our knowledge, this is the first proposed solution to this general robotics problem.

7.3 Related Work

The literature on robotic energy management has many aspects ranging from docking mechanisms, energy-efficient path-planning, to fuel types. The most relevant aspect to this work is on action-selection. Perhaps the most standard and simple way to determine when it is time for the robot to recharge is to set a fixed threshold. This can either be a threshold

[1]Webster: Czech, from *robota*: compulsory labour

directly on the energy supply as in [75] or on time elapsed since last charging as in [4]. The latter is usually easier to implement but less accurate, because one has to have some model of the energy supply. However, [93] showed a fixed threshold policy can be improved upon. While it is true that maximising the energy intake rate maximizes potential work rate, it does not optimise with respect to when the work is done. It also *assumes* that recharging is always valuable. This is often true, but not always, as we show below. Also notable is that all of the above papers refuel the robot to maximum capacity at each opportunity, and do not consider that this may not be the best policy.

[57] consider the problem of energy efficient rendezvous as an action selection problem, and so investigate the where, but not the when and how long, to refuel. [12] had robots living an a closed ecosystem learn to 'survive'. Here robots learned to choose between recharging and fighting off competitors. Birk's agents' value function of 'survivability' is different to that considered here. The rational robot and its owner are interested in gaining maximum reward by working at the robot's task, and are indifferent to the lifespan of the robot. This is a key difference between the purpose of robots and animals.

Although intended as a wake-up call for psychology research, Toda's Fungus Eater though experiment([83]) has been influential in the robotics literature. The survival quest of a mining robot on a distant planet contributed significantly to ideas of embodiment and whole agents [65], but the action selection problem presented has yet to be solved in more then the trivial way of a fixed threshold policy.

[78] and [59] investigate work - refuel cycles, or as they call it 'basic cycles', and show a simple rule, based on cue and deficit, can solve a two resource problem. The cue-deficit policy is inherently reactive and thus fails to cope with the cost of switching between behaviours, it also lacks any form of lookahead or planning, so it is difficult to see how it would handle discounted labour situations.

From McFarland's work, it is a small step into the vast literature on behavioural ecology, from which [42], [80] and [79] are strongly recommended starting points. Due to the biological background of these publications, the analogy to labour and reward in a robotic case is not obvious. The majority of this work uses dynamic programming (DP) as a means of evaluating models of animal behaviour. An exception is that does not rely on DP is [40], who investigates the biomechanics of land based animals to derive models of optimal fuel load during migration. The model optimises for migration time, does not map directly into discounted labour or the cyclic work-charge lifestyle of the long-lived robot.

It is known that animals prefer a small, immediate reward to a large delayed reward. So animals seem to do some form of time discounting. According to [48] the reason seems to be that animals in general are risk averse. From a robotics point of view, one major issue with the descriptive models of behavioural ecology is, they lack the 'how does the animal actually do it' prescriptive description and hence do not translate readily into robot controllers. Work that tries to bridge the gap between ecology and robotics is [72]. Here Seth uses ALife methods to evolve controllers that obey Herrnstein's matching law (roughly: relative rate of response matches relative rate of reward), which again is in the domain of rate maximization.

7.4 The Model

In this section we describe the behavioural model used in this work. In order to keep the analysis tractable we choose an abstract, slightly simplified model. The world is modeled

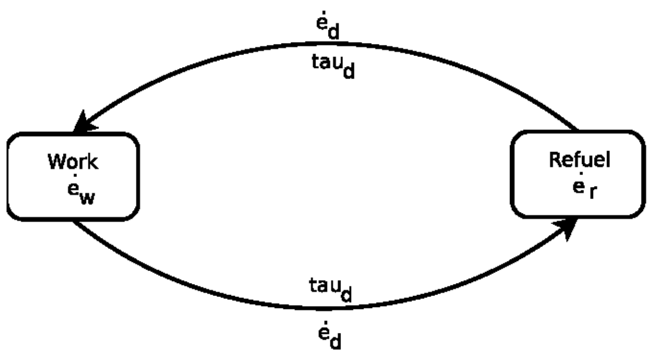


Figure 7.1: Refuel and time discounted labour model, see text for details

as two distinct spatially separated sites: a work site and a refueling site. Moving between sites has a non-zero cost (Figure 7.1). The robot has a energy storage of $E(t)$, where $0 \leq E(t) \leq E_{max}$. If the energy supply drops to zero anywhere but at the refueling site, the robot looses is ability to move or work and can gain no more reward. The robot can be in one of four states:

- refuelling with a refuelling rate of $\dot{e_r}$, to do so the robot has to be at the refuelling site.
- transitioning from the refuelling site to the work site, the duration of this transition is τ_d and the robot has to spend energy at a rate of $\dot{e_d}$, so the transitions cost in term of energy is $\tau_d \dot{e_d}$
- working, which gets the robot a reward of $R = \int_{t_0}^{t_0+\tau_w} \beta^t dt$, where t_0 is it time when the robot starts to work and τ_w is the duration the robot works for. Therefore the reward the robot earns by working is discounted with a discount factor $0 < \beta < 1$. While working the robot spends energy with a rate of $\dot{e_w}$. In other words, the robot turns energy into work and therefore reward. In case where the robot performs several work sessions, the reward is accumulated and only the overall reward is at interest to the owner of the robot. As with refuelling, work can only be performed at the work site.
- transitioning from the work site to the refuelling site, the duration of this transition is τ_d and the robot has to spend energy at a rate of $\dot{e_d}$

The robot's goal is to achieve as much reward as possible. To do so, it has to make two decisions, (1) when to stop refuelling and resume work and (2) when to stop working and refuel. We mostly refer to the action of accumulating energy as *refuelling* and not at *recharging* because we want to emphasis the general nature of our model.

It is worth pointing out that in a real world scenario all important variables, namely the energy rates, could be known in advance or are easily measured by the robot. Here we assume these variables to be constant, though in an actual implementation we would use averages as approximations. It would also be feasible to do some form of piece wise linear approximation of the energy rates. The discount factor can also be assumed to be known, since this factor is task dependent and, hence, is set by the owner or designer of the robot or by some external market. As we show below, even if all else is fixed, the robot owner can use the discount factor as a control variable that can be tweaked to fine tune the robots behaviour. Everything else is predefined by the tasks, the robot's construction or the environment.

In order to improve readability, we need to introduce some additional notation. $k_1 = \frac{\dot{e_r}}{\dot{e_w}}$ is the ratio of the energy rate while refuelling to the rate while working. Similarly, $k_2 = \frac{\dot{e_d}}{\dot{e_w}}$ is the energy rate while transitioning to the energy rate while working. $k_3 = \frac{\dot{e_d}}{\dot{e_r}} = \frac{k_2}{k_1}$ is the ratio of the energy rate while in transition and the energy rate refuelling. τ_r is the time spent refuelling during one refuel-work cycle. The amount of work the robot can perform is limited by the energy supply the robot has, so we express the potential work duration as a function of refuelling and transitioning time where $\tau_w = \tau_r k_1 - 2\tau_d k_2$, which is basically the amount of time the robot can work for, given the amount of energy the robot got from refuelling minus the energy the robot has to spend to travel to the work site and back to the charging station. We also introduce the period of time $T = \tau_r + 2\tau_d + \tau_w = \tau_r(1 + k_1) + 2\tau_d(1 - k_2)$ as the length of one refuel-work cycle.

7.5 When to stop working

Let $e(t)$ be the energy in the robot's storage at time t. At what energy level $e(t) < \epsilon_{w\to r}$ should the robot stop working and transition to the refuelling site? Since the value of work is time discounted, work that the robot performs now is always more valuable then the same amount of work performed later. This creates an inherent opportunity cost in transitioning from the work site to the refuelling site because it takes time and costs energy $\tau_d \dot{e_w}$ that cannot be spent working. This implies that the robot needs to work as long as possible now and not later. Hence the only two economically rational transitioning thresholds are:

- $\epsilon_{w\to r} = \tau_d \dot{e_d}$
  The robot stops working when it has just enough energy left to make it to the refuelling station. The robot will spend the maximum amount of energy, and therefore time, working, ensuring the highest reward before refuelling. Comparatively, should a higher transitional threshold be used, the robot would stop working earlier and refuel earlier, but discounting results in a smaller reward. Should the transitioning threshold be smaller, the robot would have insufficient energy to reach the refuelling station. In this case, the robot cannot gain any further reward because it runs out of fuel between the work and refuel sites.

- $\epsilon_{w\to r} = 0$
  The robot spends all of its energy working and terminates its functionality while doing so. At first glance this option seems counter intuitive, but one can imagine highly discounted labour situations, such as rescue missions, where the energy that would otherwise be spent on approaching a refuelling site is better spent on the task at hand. This might also be a rational option if the transition cost is very high, e.g. NASA's Viking Mars lander took a lot of energy to Mars in the form of a small nuclear reactor, rather than returning to Earth periodically for fresh batteries (the recent Mars rovers employ solar cells to recharge their batteries originally charged on Earth).

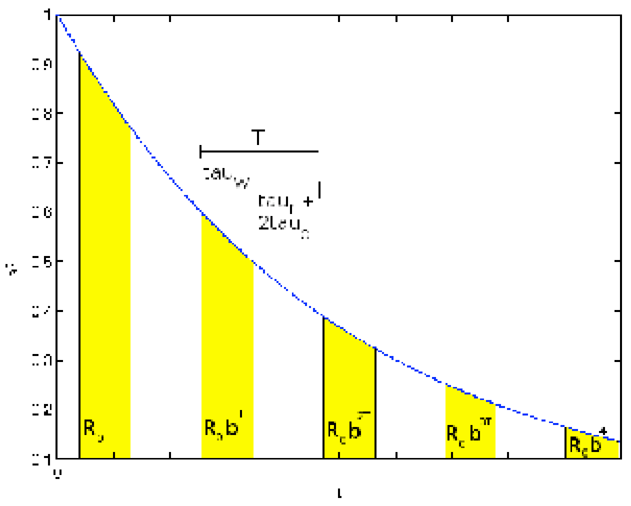


Figure 7.2: General discounting in refuel - work cycles. The shaded areas are periods in which the robot works and thus earns a reward. The white areas correspond with time in which the robot does not obtain any rewards because it either travels or refuels

7.5.1 Suicide or live forever?

Using our simple model, we can determine whether a robot in a given scenario should terminate while working or continue indefinitely with the work-refuel cycle. Let

$$R_0 = \int_{\tau_r+\tau_d}^{\tau_r+\tau_d+\tau_w} \beta^t dt = \beta^{\tau_r+\tau_d} \frac{b^{\tau_w} - 1}{\ln(\beta)} \tag{7.1}$$

be the reward obtained from spending $E_{max} - \epsilon_{w\to r}$ energy or τ_w time during the first working period (see figure 7.2). In this figure the shaded areas correspond to time in which the robot performs work and thus obtains a reward proportional to the size of the shaded area. Later work periods are discounted more strongly and hence provide a smaller reward. White areas correspond to times in which no reward is earned because the robot either travels between the work and refuelling site or it refuels. The size of this area is proportional to the opportunity cost, that is, reward that, in principal, could have been obtained if the time had been spent working.

Let T be the duration of one full work-refuel cycle, that is working - transition - refuel - transition, or $T = \tau_w + \tau_d + \tau_r + \tau_d$. Therefore, the reward gained in the next cycle is the initial reward R_0 discounted by T and becomes $\beta^T R_0$. Subsequent rewards are again discounted by T and so the reward for the third cycle is $\beta^{2T} R_0$. The sum of all rewards if working infinitely, that is choosing $\epsilon_{w\to r} = \tau_d k_2$, is

$$R_\infty = R_0 \sum_{i=0}^{\infty} \beta^{iT} = R_0 \frac{1}{1-\beta^T} \tag{7.2}$$

In practise no system will last forever, so this analysis is slightly biased towards infinite life histories.

If the robot chooses $\epsilon_{w\to r} = 0$ it gains the initial reward R_0 plus a one time bonus of

$$R_+ = \int_{\tau_r+\tau_d+\tau_w}^{\tau_r+\tau_d+\tau_w+\tau_d k_2} \beta^t dt = \beta^{\tau_r+\tau_d+\tau_w} \frac{\beta^{\tau_d k_2} - 1}{\ln(\beta)} \tag{7.3}$$

by spending the energy required for transitioning on working. The reward gained over the live time of the robot (which is fairly short) is $R_{rip} = R_0 + R_+$.

So the answer to Q2 is that the rational robot selects that threshold $\epsilon_{w\to r}$ that achieves the higher overall reward, so it picks

$$\epsilon_{w\to r} = \begin{cases} 0 & : \quad R_{rip} \geq R_\infty \\ \tau_d \dot{e_d} & : \quad R_{rip} < R_\infty \end{cases} \tag{7.4}$$

Since the discount function $\int \beta^t dt$ belongs to the class of memory-less functions, we only have to calculate eq. 7.4 once, in other words if it is the best option to refuel after the first work cycle it is always the best option to do so and vice versa.

7.6 How much energy to store

We have shown how to determine a threshold for transitioning from work to refuelling. In this section we will analyse when to stop refuelling and resume work, or phrased differently, how much energy to accumulate before starting to work. Energy and time are interchangeable elements, provided that we know the rate at which energy is spent and gained. Since discounting is done in the time domain, our analysis equates energy with time for simplicity. Based on this, we can ask the time equivalent of Q1: 'how long should the robot refuel for?' We call this refuelling duration τ_r. To be rational, the robot must refuel long enough to gain enough energy to make the trip to the work site and back, that is $2\dot{e_d}\tau_d$, otherwise it would have to turn around before reaching the work site and thus will not gain any reward. Refuelling after the storage tank is full is time wasted that would better be spent obtaining a reward. Therefore the refuelling time is limited to $2\tau_d k_3 \leq \tau_r \leq E_{max}\dot{e_r}^{-1}$. In the following we assume, without loss of generality, the robot starts at the refuelling site with an empty fuel tank. Assuming differently will just result in shift of the analysis by a constant factor, but will not change the overall conclusions.

7.6.1 Acyclic tasks

First we examine situations in which the robot has to refuel for a task that has to be done only once, that is the robot refuels, performs the task, and returns to the refuelling site. Depending on the time spent refuelling the robot obtains the following reward during the upcoming work period.

$$R(\tau_r) = \int_{\tau_r+\tau_d}^{\tau_r+\tau_d+\tau_w} \beta^t dt = \beta^{\tau_r+\tau_d} \int_0^{\tau_r k_1 - 2\tau_d k_2} \beta^t dt \tag{7.5}$$

Next we need to find the $\hat{\tau_r}$ that maximises $R(\tau_r)$, which is

$$\hat{\tau_r} = argmax(R(\tau_r)) = \frac{\log_\beta\left(\frac{1}{k_1+1}\right) + 2\tau_d k_2}{k_1} \tag{7.6}$$

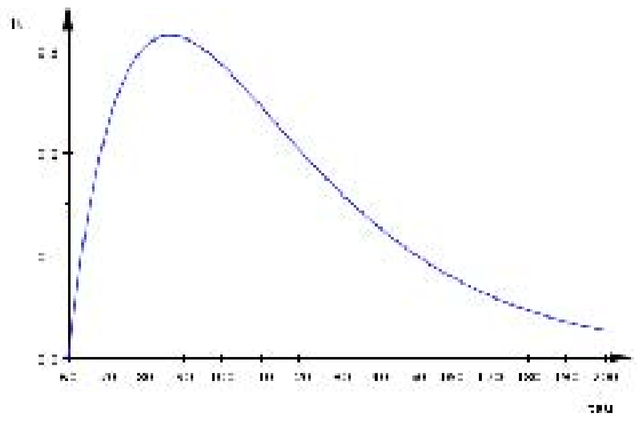


Figure 7.3: Reward depending on refuelling time with an example configuration $k_1 = 0.5, k_2 = 0.5, \beta = 0.97, \tau_d = 85s$

Figure 7.3 shows an example reward function (eq. 7.5) depending on the refuelling duration τ_r. Using eq. 7.6 we calculate that for this particular configuration the reward is maximised when the robot refuels for $\hat{\tau}_r = 86.6234\ldots$. If the fuel tank is filled before that time, the best the robot can do is return to work. This will give it the highest reward achievable, but the designer should keep in mind that there might exist a class of robots with a larger fuel tank that will achieve a higher reward. Note that if the robot stops refueling at $\hat{\tau}_r$ even if its energy store is not full to capacity, and transitions to working, it earns the highest reward possible. To our knowledge this has not been stated explicitly in the robotics literature before. It is generally assumed that robots should completely recharge at each opportunity, but this is not always the optimal strategy.

7.6.2 Cyclic tasks

In cyclic tasks a robot is required to always return to work after resupplying with energy. Here the analysis is slightly different then in the acyclic case because the refuelling time of the *current* cycle not only influences the duration and length of the work period of this cycle but of all cycles to come. Hence, we should select a refuelling threshold that maximises the overall reward. The overall reward is calculated by (see fig. 7.2)

$$R_\infty(\tau_r) = R_0 \sum_{i=0}^{\infty} \beta^{iT} = \frac{(\beta^{\tau_w} - 1)\beta^{\tau_r + \tau_d}}{(1 - \beta^T)\ln(\beta)} \tag{7.7}$$

Unfortunately, it seems impossible to find a closed form solution to $\hat{\tau}_r = argmax(R_\infty(\tau_r))$. However, eq. 7.7 can easily be evaluated for a given τ_r and so calculating the reward for each of a finite set of values for τ_r and selecting the one that maximises the reward is quite practical. In any real application the number of τ_r to be tested is limited and possibly rather small, in the order of a few thousand. This is because any real robot will have a finite energy storage and any practical scenario will require only limited sampling due to the resolution of the fuel gauge, the uncertainty in the environment, etc. In the case of our Chatterbox Robot (see below), the battery capacity is 2.8 Ah and the fuel gauge has a resolution of 1mA, resulting in less then 3000 calculations for an exhaustive search.

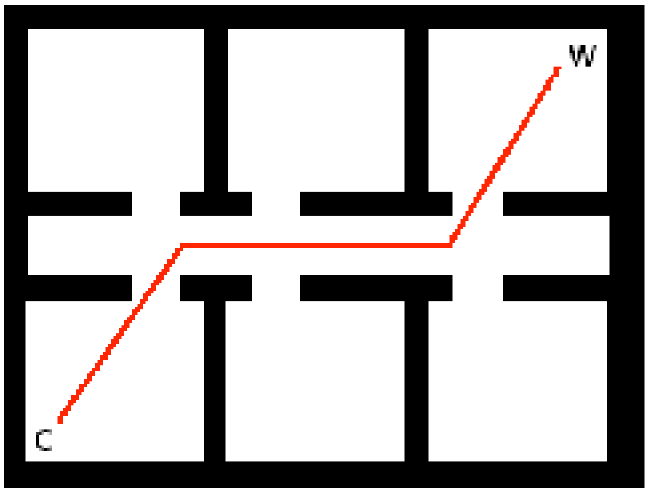


Figure 7.4: Office like environment with a charging station 'C' and a work site 'W'. The red line is the stylised path the robot travels on.

7.7 Experiments

In this section we present experiments to validate the theoretical results described in detail above. All experiments were performed using the robot simulator Stage[2]. The simulated robot uses simulated electrical energy, where we assume charging and discharging to be linear, with constant current for charging I_c, working I_w and driving I_d. We further ignore any effects caused by the docking mechanism, change in battery chemistry or ambient temperature.

In all experiments we roughly model a Chatterbox robot, a robot designed and build at SFU based on an IRobot ICreate platform. This robot has a battery capacity of approximately 2.8Ah and draws about 2A while driving. We defined an abstract work task which consumes 4A of current. Once at a charging station, the robot docks reliably and recharges with 2A. The world the robot operates in is office-like with one charging station and one work site shown in fig. 7.4. The obstacle avoidance and navigation controller drives the robot from the charging station to the work site and vice versa in approximately $\tau_d = 85s$. Due to naturally accruing noise in the experimental setup the travel time may vary by up to 6 seconds. While working, the robot receives one unit of reward per second, discounted by β. Discounting occurs on a one second basis.

7.7.1 Cyclic Task

The goal of this experiment is to evaluate how closely our analysis from section 7.6.2 matches a robot in a simulated environment. In this experiment the robot's task is to recharge for some time τ_r, proceed to the work site, work until the battery energy drops to $\epsilon_{w \to r} = \tau_d I_d$, return to the charging station, and repeat the process. The reward for work is discounted by $\beta = 0.9997$. To find out which τ_r maximises the reward we varied the threshold for leaving the charging station $\epsilon_{r \to w} = \tau_r I_c$ in each try. A trial lasted for 50000 seconds ($\approx$ 13.8 hours).

[2]http://playerstage.sourceforge.net

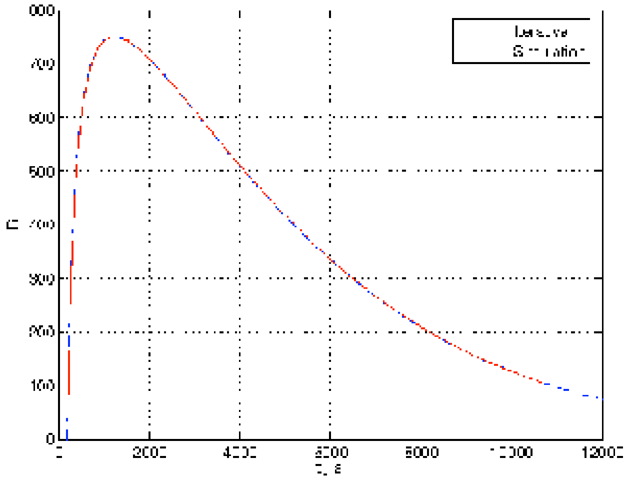


Figure 7.5: Comparing analytical and simulation results for accumulated reward from a **cyclic** task depending on refuelling time with an example configuration $I_c = 2.0, I_d = 2.0, I_w = 4.0, \beta = 0.9997, \tau_d \approx 85$

Figure 7.5 compares the accumulated reward gained over τ_* from the simulation and from the best solution obtained from the model by iterating over eq. 7.7. The recharging time that maximises the reward is predicted by the model to be $\hat{\tau}_* = 1219$ and in the simulation $\hat{\tau}_* = 1170$. The difference comes from the variation in time, and therefore energy, the robot requires to travel between the charging station and the work site. Not only does this change the starting time of work which influences the reward, it also makes it necessary to give the robot a small amount of spare energy to ensure it would not run out of battery. This, in turn, delays charging and thereby influences the reward gained. However, the empirical results agree qualitatively with the values predicted by the model, and the optimal recharging time predicted by the model was within 4% of that observed in the simulation.

7.7.2 Acyclic Task

As before, we perform this experiment in order to compare the theoretical results with a simulation. The setup is the same as in the cyclic task experiment with the difference that the robot only has to perform one charge-work cycle. Figure 7.6 compares the simulation results to the analytical results. Where the general shape of the curve is similar to that in the cyclic task, it is worth to point out that the maximum reward is gained with a larger charging threshold. This is intuitively correct as the robot has only once chance to obtain a reward. It can be (depending on the discount factor) beneficial to begin work later, but to work for a longer period. For our configuration, the most profitable theoretical charging time is $\hat{\tau}_* = 2872.7$ and the best simulation results were obtained with $\hat{\tau}_* = 2880$. Again the difference between the theoretical and experimental results, barely visible in the plot, are due to imprecision in the robot simulation.

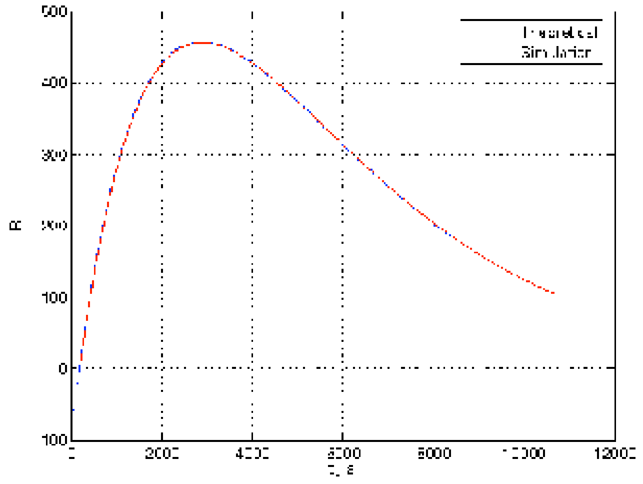


Figure 7.6: Comparing analytical and simulation results for accumulated reward from an **acyclic** task depending on refuelling time with an example configuration $I_c = 2.0, I_d = 2.0, I_w = 4.0, \beta = 0.9997, \tau_d \approx 85$

7.7.3 Once or forever

In a further experiment we investigate the circumstances under which it is more profitable, and hence rational, for the robot to fully deplete its energy supply while working and when it is better to choose a perpetual refuelling policy. As in the previous experiments we use a simulated Chatterbox robot with the previously described parameters in the office-like environment. For this scenario, we vary the discount rate between 0.9850 and 0.9999 in 0.0005 increments and run two sets of simulations. In the first, the robot depletes it's energy supply while working, that is, we choose the leave work threshold $e_{w \to *} = 0$. For the second set, we choose $e_{* \to w} = \tau_d \dot{e}_d$, a leave work threshold that causes the robot to keep performing work-refuel cycles forever. Since we change the discount rate we have to adapt the leave refuelling site threshold in order for the robot to earn the highest possible reward. For this determine the optimal threshold in the same way as for the previous experiments. Figure 7.7 depicts the rewards obtained for different discount factors with each policy. As the graph further shows, for higher discounting (smaller discount rate), it is beneficial for the robot to choose a one time work policy. Conversely, for smaller discounting (higher β), it pays to keep working. The theoretical discount rate for switching the policy from one work period to an infinite work refuel cycle is $\beta = 0.9979$, which, as the graph shows, closely resembles the experimental result.

7.8 Discussion and Conclusion

We outlined a theoretical analysis of when to refuel and for how long to refuel a robot in situations where the reward for the robot's objective is discounted over time. This discounting is, more often then not, ignored in robotics literature, although it is at the very base of rational behaviour ([81]). We took theoretical results and demonstrated that they apply to a simulated robot. In these simulations we assumed the location of and the distance

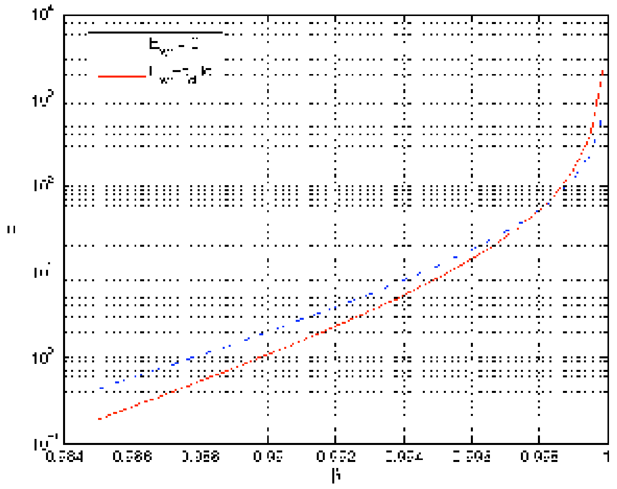


Figure 7.7: Reward obtained for different discount factors with two leave work thresholds. Configuration $I_c = 2.0, I_d = 2.0, I_w = 4.0, \tau_d \approx 85$

between work and refuelling station to be known. This is reasonable in the state of the art in mapping and localization, in a wide range of scenarios. We further assumed the average energy spending rates to be constant and known, something achievable in most cases. One assumption made that simplifies a real-world robot scenario is the refuelling rate. Gasoline-powered vehicles which refuel from a standard gas station have a constant refuelling rate, or close to it. However, the charging rate of a battery may depend on many factors including the charging method used, temperature, battery chemistry, and the current capacity of the battery. One useful extension of our model would be to include a realistic chemical battery recharge transfer function.

This paper has presented and analyzed a core action selection problem for autonomous agents such as animals and mobile robots: how much to fuel before working, and when to abandon working and return to fueling, such that the value of discounted work is maximized. A simple model readily provides answers to these questions and closely predicts the observed behaviour of a robot simulation. Whille the model is simple, it is very general, and these results suggest that it could be of practical as well as theoretical interest. We propose it as a baseline to build upon.

Chapter 8

Adaptive multi-robot bucket brigade foraging

Authors: Adam Lein and Richard Vaughan

8.1 Abstract

Bucket brigade foraging improves upon homogeneous foraging by reducing spatial interference between robots, which occurs when robots are forced to work in the same space. Robots must spend time avoiding one another instead of carrying out useful work. Bucket brigade foraging algorithms restrict the motion of each robot to at most some fixed distance from its starting location. We reproduce the performance of known bucket brigade foragers, and then present a new controller in which robots adapt the size of their foraging area in response to interference with other robots, improving overall performance. This approach also has the potential to cope with nonuniform resource distributions.

8.2 Introduction

The *foraging problem* is a task in which one or more agents must find and collect target objects and deliver them to a "home area". The simplest adaptation of this problem to multi-robot systems is for many robots to independently solve the foraging task, a solution known as *homogeneous foraging*.

[74] explore foraging strategies in large-scale multi-robot systems. The key variable in their experiments is spatial interference: destructive interactions between robots forced to perform work in the same space. Multi-robot systems are advantageous only when what might be called the marginal performance – the benefit to performance of adding a single robot to the system – is positive. However, as those authors demonstrated, there comes a time when this is no longer so, when the loss of performance due to interference between robots outweighs the gain in work done by having more workers.

[35] and [63] describe *bucket brigading*, in which each robot is restricted to a finite search area, and instead rely on their coworkers to both deliver pucks into their search area, and remove pucks out of it. By restricting each robot to a finite area whose size is

determined a priori, interference is ameliorated. [74] empirically investigate the performance of large groups of robots as a function of varying search area sizes; homogeneous foraging corresponds to search areas of infinite radius. [63] describe the expected performance of multi-robot, space-constrained systems as a curve to which the $R = 40m$ curve in Figure 8.2 roughly corresponds; the curve has a local maximum, after which the marginal performance is negative.

In this research, we investigate an approach to bucket brigading that does away with a priori search radii by allowing robots to adapt their search radii in response to interference with other robots, improving overall perfomance.

8.3 Related work

Foraging is a common analogy for a wide variety of robot tasks, including exploration and mapping, search, and actual objects retrieval.

[93] applied the rate-maximizing foraging model to a single robot performing the task of foraging over a long period of time. The robot had a finite energy supply, and was required to travel to a charging station to recharge its batteries. While recharging, and while traveling between the work site and the charging site, the robot is not doing work. The authors presented a scalable, online, heuristic algorithm for the robot to recharge efficiently, maximizing the proportion of its time it spends working.

In [103], building on [16], the problem of spatial interference in multi-robot systems was addressed through the use of aggressive display behaviors. Several robots were required to perform a transportation task (akin to our foraging task) in shared space. Robots selected an "aggression level" based on the amount of work they had invested up to that point. The discrepancy in aggression levels between interfering robots was used to break the symmetry that would otherwise have lead to deadlock. The authors showed their approach to be effective, both in simulation and in a real-world implementation.

[53] formally modeled the effect of interference on the performance of a swarm of foraging robots. Their model formulated as a system of coupled first-order nonlinear differential equations. They found that group performance grows sublinearly with group size, so that individual performance actually decreases with increasing group size. Simulations verified the predictions of their model.

[69] performed experiments in which real robots perform a foraging task using a variety of simple communication methods. Robots communicated by flashing a light bulb under various circumstances. The authors showed that communication can reduce the variance in the robots' performance. In contrast, this research does not use explicit communication between robots; communication is implicit, however, in that robots must alter their behavior in the short term in response to the presence of other robots (collision avoidance), and in the long term by adapting a parameter of their behavior (discussed later).

8.4 Simulation

In order to reproduce the results of [74], and compare them to the performance of the adaptive bucket brigaders, we developed a simulator similar to that in [74]. A description follows:

Robots are located on a 64 meter-square plane scattered with pucks, in the northeast corner of which plane is a 3 m quarter circle "home area". Robots are equipped with grippers which they can use to retrieve these pucks, and when a puck is dropped off in the home area, it is said to have been foraged.

Robots can move forward at a rate of 0.1 m/sec, and can turn to either side at a rate up to once every five seconds. It takes four seconds to retrieve a puck, but a carried puck can be dropped instantaneously.

Robots can sense walls and other robots within 0.6 m of their centers without error (compared with 0.5 m in [74]), via the use of 12 radially oriented sensors. They cannot sense pucks unless the puck in question is located directly within the grip of their grippers, and this sensing is also without error. Robots can determine the direction towards the home zone at any time; [74] explain this as the use of a "four-bit compass". Robots use the data from their proximity sensors to avoid walls and other robots.

[74] adds a variety of noise to each parameter in their simulation. For simplicity, we have dispensed with this noise, except where explicitly mentioned.

8.4.1 Parametrized bucket brigading

The purpose of the overall robot system is to retrieve pucks and deliver them into the home area – to forage the pucks. In the bucket brigade approach to this problem, individual robots do not attempt to carry a puck all the way to the home zone themselves, but rather merely to shift the distribution of pucks towards the home area.

Each robot will attempt to stay within a zone, the size and location of which it keeps track. The robot can determine how far it is from the center of this zone, and can tell the direction towards the center of the zone. However, this memory is noisy; the zone within which any robot stays drifts on a random walk at a rate of 0.01 m/sec.

The robot searches via a naïve algorithm within its zone. If it ever leaves the zone, it will drop of any puck it may be carrying, return towards its zone, and continue searching. If it ever discovers a puck, it will retrieve it and head towards the home are. The effect is that a "brigade" of similarly-behaving robots will slowly transport pucks from one robot (zone) to the next, gradually bringing the puck closer to the home area. Of course, ultimate delivery of the puck requires a connected sequence of overlapping zones ending in the home area, but this may be achieved over time (even if never simultaneously) due to the drift of the zones.

Shell and Matarić's robots all share the same ranges. In the following sections, we will explore other approaches to assigning ranges to robots. In any given experiment, every robot uses the same approach to range selection.

8.4.2 Radial parametrization

Given that the emergent behavior of the robots is to shift the distribution of the pucks towards the home area, the first natural modification to the controller would be to allow the range parameter of robots to vary with the distance to the home area. Effectively, we are discarding the assumption that the distribution of pucks is uniform, but supposing that they may be more densely distributed near the home area, and therefore a different range parameter would be optimal. To demonstrate this, we simulated robots whose range parameter varied linearly with distance from the home area.

Let us consider the following idealized, one-dimensional situation: robots and pucks sit uniformly distributed on a line of length L, with the home zone at one end. Every robot collects a puck at the same time, drives a fixed distance D towards the "home" end, deposits

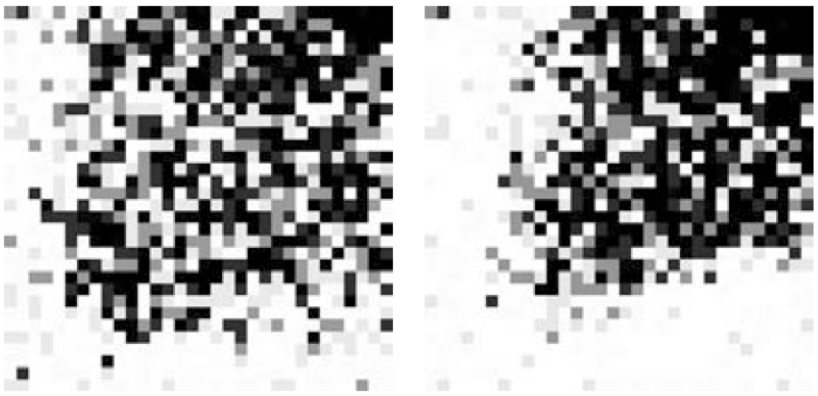

(a) (b)

Figure 8.1: Typical density distribution of pucks after (a) 40 minutes and (b) 80 minutes. One square is 2 m $\times$ 2 m; darker squares indicate a higher concentration of pucks. The home area is in the northeast corner of the world. In this simulation, pucks were not added to the world after the start of the experiment.

its puck, and then returns to its starting location, at which point the process repeats. Let $p_n(r)$, where r is the distance from a given point to the home end, be the density of pucks at that point after n of these cycles has taken place. $p_0(r)$ is some constant (the initial puck density). Then

$$p_{n+1}(r) = \begin{cases} (1-c)p_n(r) & \text{if } r > L - R, \\ (1-c)p_n(r) + cp_n\left(\frac{Lr}{L-D}\right) & \text{otherwise.} \end{cases} \tag{8.1}$$

In our simulation, an analogous process occurs in two dimensions. Puck densities at two time-steps are displayed in Figure 8.1 during a typical run of an experiment in which pucks are not added to the system as fast as they are removed from the system by foraging (pucks in the other experiments in this work are added just as fast as they are foraged, to maintain constant average puck density).

In all experiments, pucks are initially distributed at random. However, it can clearly be seen that as soon as the robots interact with the pucks, the distribution becomes less random, biased toward the home area — the system has a form of entropy that decreases as a result of the work of the robots. Since the optimal search radius for a robot depends on the density of pucks, once the density of pucks changes, the robot's original choice for search radius may no longer be optimal. Ideally, robots would know and select the optimal choice for search radius on an ongoing basis, but in these experiments, robots are not given enough information to achieve this ideal. In the next section, we propose a method for approximating this ideal.

8.4.3 Adaptive range selection

The aforementioned ideal puts an extra burden on the robot — it must be constantly aware of the distance between the center of its zone and the center of the home area. Alternatively, it must be able to measure the local puck density. In place of these assumptions we may also allow robots to adaptively select their range parameter using purely local information.

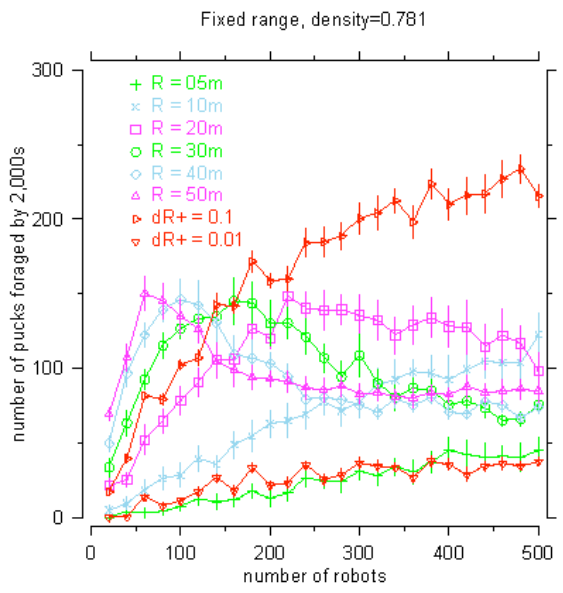


Figure 8.2: Performance with fixed and adaptive search radii at 0.781 pucks/m^2. R is the search radius of each robot in the trial. The curves labeled dR^+ show the results for adaptive range selection.

In *adaptive range selection*, a robot will continuously increase its range parameter at rate dR^+, except while it is avoiding a collision with another robot, to which situation the robot will react by shrinking its zone at rate dR^-, thus making it less likely to interfere with other robots in the future.

Results for adaptive range selection are included in Figure 8.2 and 8.3.

8.4.4 Experimental design

Initially, we followed [74] in experimental design. The following parameters were varied: puck density (0.781/m^2 and 3.125/m^2), search area radius (5, 10, 20, 30, 40, or 50 m), and number of robots (20, 40, 60, 80, ..., 500). The task was simulated for each combination of parameters for 2000 simulated seconds, and the number of pucks foraged after that time was recorded. Twenty such trials were run, each with a different initial distribution of pucks; to control for robot position, robots were initially placed on a square lattice. The reported results, in Figures 8.2 and 8.3, are the averages of those twenty trials. Error bars indicate the standard deviations of the twenty-trial experiments.

Next, we tested our adaptive range selection controller using the same experimental setup. For these experiments, search radii were allowed to increase by $dR^+ = 0.1$ m/s and decrease by $dR^- = 0.05$ m/s, biasing the robots towards the limit of homogeneous foraging in the absence of significant interference. Each robot began with a small range of 5 m. For each parameter set, twenty trials were run, and mean performances were plotted with standard deviations shown.

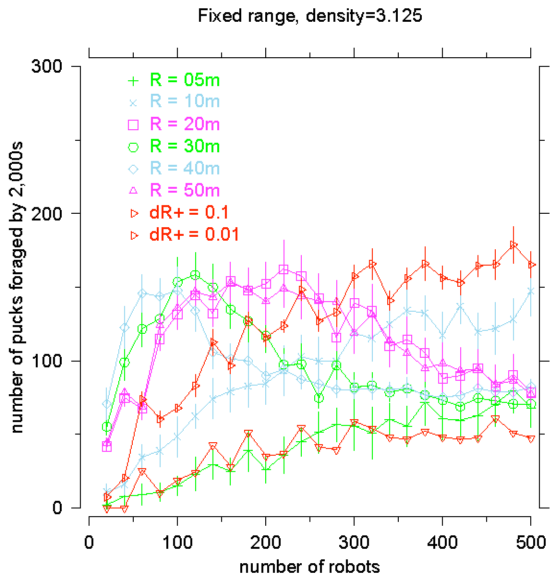


Figure 8.3: Performance with fixed and adaptive search radii at 3.125 pucks/m^2. R is the search radius of each robot in the trial. The curves labeled dR^+ show the results for adaptive range selection.

8.5 Results

Our first step was to reproduce the results in [74]. Visual analysis of the data in Figures 8.2 and 8.3 indicates that this was accomplished, in that increasing the radius of robots' search areas in the fixed-radius regime led to an increase in the marginal benefit of adding robots (i.e., to the slope of the curves in those graphs), but only up to a point: eventually, adding more robots decreases the performance of the system as overcoming interference begins to dominate the robots' behavior. These critical points are clearly visible as the significant local maxima in the $R = 30$ m, $R = 40$ m, and $R = 50$ m curves.

There is a noteworthy distinction in that robots in our experiments only foraged approximately half the pucks that those of [74] did; this indicates more a quantitative difference in the efficacy of the robots' controller programs than a qualitative failure of our simulations to agree at the heart of the matter: that interference affects a growing population later when the individuals' foraging spaces are larger, which is indicated by the relative shapes of the curves.

Our adaptive range selection algorithm performed at least as well as the fixed ranged controllers in simulation, and scaled better. Fixed range algorithms suffered from one of two problems: robots with small search areas did not gather many pucks, and robots with large search areas interfered too much and the critical point at which the marginal benefit of increasing the number of robots was reached when the number of robots was still small. While the robots using adaptive range selection did not gather as many pucks when the number of robots was small as did robots with large, fixed search radii, increasing the number of robots always increased the performance of the group.

Also noteworthy is that the adaptive controllers performed more consistently, as indicated by tighter error bars on those curves than on the fixed-radius performance curves. Since adaptation to interference is a form of implicit communication, this is in agreement with the findings of [69].

Adaptive range selection was sensitive to variations in dR^+ and dR^-. If dR^+ was too small, adaptive selection underperformed the fixed range foragers. Figures 8.2 shows results when $dR^+ = 0.01$, a fifth of dR^-. In that case, the adaptive controller performs no better than the worst-performing fixed-range controller we tested. This is not altogether surprising, since the search radius in the worst fixed-range controller and the initial search radius in the $dR^+ = 0.01$ m adaptive controller were both 5 m.

Figure 8.3 shows qualitatively similar improvements; however, in this scenario (where puck density is 3.125 pucks/square meter, the improvement is only slight, even for large group sizes).

8.6 Future work

In this paper, we have explored only some of the ways — and naïve ones at that — of improving on the bucket brigading algorithm. Future work may explore generalization of the problem to discard hidden assumptions, or, increasing the adaptability of the system. For instance, a more plausible biological analog might be ants foraging for food. In such a scenario, the ants would not initially be uniformly distributed through the field, nor would the food.

The performance of the adaptive-parameter bucket brigade forager needs to be more deeply tested. In none of our experiments did we note negative marginal performance; the limits of the algorithm in terms of scalability remain to be seen. The parameter space for the adaptive controller — the possible values of dR^+ and dR^- — needs to be explored in greater depth. We should also test the controller in more topologically complex spaces, such as corridors, as did [63], and investigate why the adaptive controller does not outperform as significantly in denser environments.

Briefly mentioned above is the point that the adaptive approach causes a net increase in the number of a priori parameters: new parameters added are the rates at which zones increase and decrease in size. In this work, those parameters were set experimentally. Future work may provide motivation behind values of these parameters, investigate a relationship between optimal values for dR^+ and dR^-, or do away with these parameters entirely.

Future work may also formalize a closed-form relationship among optimal search radius, puck density, and robot density. Additionally, we have briefly touched on a notion of entropy in robots' work space — a quantitative function of the environment that decreases through *useful* interactions between agents and the environment — this notion should be more formally analyzed.

8.7 Conclusions

To summarize the contributions of this paper: we replicated the results of [74], confirming their findings with an independent implementation. Further, we propose a simple modification of their foraging scheme in which each robot's foraging area is adapted in response to interference. The new method was shown to improve performance, particularly in large population sizes.

Chapter 9

Massively Multiple Robot Simulations in Stage

Author: Richard Vaughan

This chapter is based on a paper accepted for publication by the journal *Swarm Intelligence* in March 2008. The work reported was not funded by this contract, but the information here is relevant to the project so it is included for convenience.

9.1 Abstract

Stage is a C++ software library that simulates multiple mobile robots. Stage version 2, as the simulation backend for the Player/Stage system, may be the most commonly used robot simulator in research and university teaching today. Development of Stage version 3 has focused on improving scalability, usability and portability. This paper examines Stage's scalability.

We propose a simple benchmark for multi-robot simulator performance, and present results for Stage. Run time is shown to scale approximately linearly with population size up to 2,000 robots. For example, Stage simulates 1 simple robot at around 1,000 times real time, and 1,000 simple robots at around real time. These results suggest that Stage may be useful for swarm robotics researchers who would otherwise use custom simulators, with their attendant disadvantages in terms of code re-use and transparency.

9.2 Introduction

Stage is a C++ software library that simulates multiple mobile robots. Stage version 2, as the simulation backend for the Player/Stage system, may be the most commonly used robot simulator in research and university teaching today. The author is the originator and lead developer of Stage. The development of Stage version 3 has focused on improving scalability, usability and portability. This paper examines Stage's scalability.

This paper is the first description of using Stage for massively multi-robot experiments, suitable for swarm robotics and other research where the behaviour of large robot populations is of interest. In a previous paper we described how an earlier version of the Player/Stage system was useful for experimenting with a few tens of simulated robots, with controllers transferring easily to real robots [31]. Here we focus on using Stage stand-alone (i.e. without Player) as a platform for much larger robot experiments. To help researchers decide whether Stage will be useful for their project, we propose a simple benchmark for multi-robot simulator performance, and present results for Stage in a range of simulation scenarios and population sizes. This paper introduces the current development version of Stage, which will be released as Stage version 3.0.0 This version has been substantially rewritten since the last release and published description. While it provides similar functionality to version 2, it is considerably faster.

Run time is shown below to scale approximately linearly with population size up to at least 2,000 simple robots. For example, Stage simulates 1 simple robot at around 1,000 times real time, and 1,000 simple robots at around real time. These results suggest that Stage may be newly useful for swarm robotics researchers who would otherwise be using in-house custom simulators, with their attendant disadvantages in terms of code re-use and transparency.

9.2.1 Technical and methodological features

Stage's most important technical features are (i) its cooperation with Player[34], currently the most popular robot hardware interface, which allows offline development of code designed for real robots; (ii) it is relatively easy to use, yet (iii) it is realistic *enough* for many purposes, striking a useful balance between fidelity and abstraction that is different from many alternative simulators; (iv) it provides models of many of the common robot sensors; (v) it runs on Linux and other Unix-like platforms, which is convenient for most roboticists; and (v) it supports multiple robots sharing a world.

Stage also has some very important non-technical features: (i) Free Software license, the GNU General Public License version 2[1]; (ii) active community of users and developers worldwide; and (ii) status as a well-known platform. The positive feedback effect of these last two features is very important, as it creates the potential to improve research practice by encouraging replication of experiments and code reuse.

The increased performance of Stage now makes it potentially interesting to researchers who may have previously rejected using a general-purpose simulator due to lack of scalability. Using Stage (or a derivation of it) instead of building a custom simulator can save researcher time and money. But perhaps more important is the methodological advantage of using a well-known, free and open platform, with open source code, in that it encourages transparency, replication and modification of experiments - a vital part of the scientific method that is uncommon in our field due to the difficulty and cost of reimplementation.

9.2.2 Scope and outline

This paper is not intended to be a definitive guide to Stage's internals, nor a user guide, nor a comprehensive review of the state of the art in robot simulation. The aim is to concisely present the key features of Stage relevant to the swarm robotics community, including the first documented performance data and the public introduction of the Stage API.

The paper proceeds as follows: first we present some evidence for the claims made in this introduction. Next we identify the current alternatives to Stage. Then we describe the key

[1]`http://www.gnu.org/licenses/old-licenses/gpl-2.0.html`

implementation details of the simulation engine that are relevant to large-population performance. Section 9.7 describes a minimal and prototypical "swarm robotics" robot controller and set of simulation environments for benchmarking simulator performance. Section 9.8 provides benchmark results for Stage, followed by ideas for further improving performance and new features.

9.3 Evidence of Stage Impact

It is not practical to make a direct estimate of Stage installations or usage, as Stage's Free Software license allows free use and distribution. Most alternatives to Stage have distribution terms that make their use similarly uncountable, so we can not know their relative popularity. Yet Stage has a presence at the major research conferences and on university course web pages that suggests it is widely used and useful. For example, at least one paper using Stage for multi-robot simulations has appeared at ICRA, the main annual robotics meeting, every year since its debut in 2001 [97, 29, 8, 73, 9, 55, 7, 19, 17]. Only the first of these papers is by an author of Stage.

Player/Stage has been proposed as as a "unifying paradigm to improve robotics education" [2] and is used for classes at many universities (by professors who are not P/S authors) including Georgia Tech's CS3630, The University of Delaware's CISC 849, Washington University at St. Louis's CSE 550A, Brown University's CS 148, the University of Tenessee's CS 594, and the University of Southern California's CSCI 584[2]

The "official" Stage distribution is maintained by the author at the Player Project website, hosted at SourceForge [3]. Sourceforge tracks download statistics for each package, and the download history for Stage is shown in Figure 9.1. Due to occasional failures of the statistics service, these numbers are (presumably slight) underestimates. Stage was downloaded from Sourceforge at least 22,121 times between December 2001 and November 2007. Monthly downloads have grown steadily to a current rate of around 700 per month.

9.4 Related Work

A good, detailed comparison of multi-robot simulators is overdue. A recent, flawed[4] short survey paper discussed some popular systems available in 2007. [22]. We do not have space for a survey here, but the reader should be aware of the following influential systems.

- **TeamBots:**[5] This simple 2D Java-based multi-robot simulator was in popular use around 2000, due to its ease of use and free distribution terms, and the ability to run the same code in simulation and on real robots. One of the original Player/Stage goals was to reproduce the good features of TeamBots without the dependence on Java. TeamBots is still available, but development appears to have stopped in 2000 [5].

[2]Web pages for all these courses, describing assignments using Player/Stage, were available online at the time of writing (November 2007).

[3]`http://playerstage.sourceforge.net`

[4]Stage and Gazebo, two of the best known simulators, with different goals, scope, technology, scaling characteristics, authors and user community, are conflated into one score, and surprisingly compared with Microsoft Flight Simulator [22].

[5]`http://www.teambots.org`

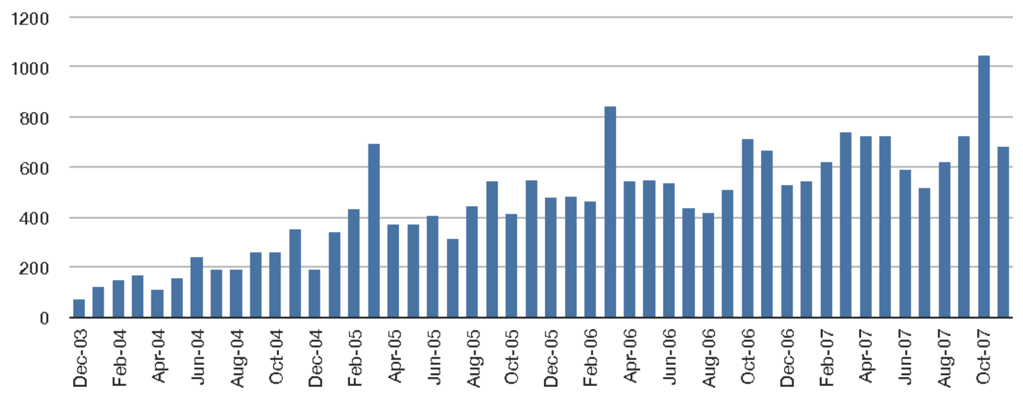


Figure 9.1: Stage downloads from Sourceforge by month, December 2003 to November 2007 (total 22,121).

- **Gazebo:**[6] A 3D simulator with dynamics from the Player Project, Gazebo is based on the Open Dynamics Engine[7]. Gazebo has become popular as a powerful, high-fidelity simulator that works with Player and has a GNU GPL license. However, it runs slowly with large populations (though no benchmarks are published) [49].

- **USARSim:**[8] Using the "Unreal" video game engine, USARSim is a GPL licensed 3-dimensional simulator that is similar in scope to Gazebo, Webots and Microsoft Robotics Studio. An important contribution of USARSim is its models of the NIST reference environments used for the RoboCup Urban Search And Rescue competition. Compatible with Player and MOAST [71], USARSim is under active development and has an active user community. USARSim aims for accurate dynamic models, and has been shown to support large worlds, though no data are available for large robot populations [18].

- **Webots:** [9] This very high-quality commercial simulator focuses on accurate dynamical models of popular robots. It is fast (though no benchmarks are published), easy to use and has an attractive interface. Webots has a "Fast2DPlugin" extension that is optimized for simple, fast simulations, and comes with detailed models of robots popular in swarm robotics such as the Alice, Khephera and E-Puck robots, plus RoboCup soccer league models. Unfortunately no scaling data are available, and Webots is not free to distribute or modify [60].

- **Microsoft Robotics Studio:**[10] Released in early 2007, Microsoft Robotics Studio is functionally very similar to Player and Gazebo, though it works only on Microsoft's Windows platform. Microsoft is promoting and financially supporting the use of Robotics Studio at some universities, notably Georgia Institute of Technology

[6]`http://playerstage.sourceforge.net/index.php?src=gazebo`
[7]`http://www.ode.org`
[8]`http://usarsim.sourceforge.net`
[9]`http://www.cyberbotics.com`
[10]`http://msdn2.microsoft.com/en-us/robotics/default.aspx`

and Bryn Mawr College through the Insitute for Personal Robots in Education [11]. Like Gazebo and Webots, Robotics Studio is based around a high-fidelity dynamics engine. Currently no examples of large population sizes seem to be available, but it is likely, provided Microsoft continues to support the project, that this system will be increasingly used in the coming years. Robot studio is free to use, but not to modify or distribute, and the source code is not publicly available.

- **Swarmbot3d (S-bot simulator):**[12] The SWARM-BOT project and s-bot robot system by Mondada, Dorigo and colleagues is a highly successful and influential swarm robotics project. The project has its own **swarmbot3d** simulator, which necessarily includes dynamics as the s-bots interact with rough terrain and by pulling on each other. The simulator is possibly the most sophisticated considered here and is based on the "Vortex" commercial physics engine. Swarmbot3d is described in detail in [62], including performance data for models at various levels of detail. It appears to run at speeds comparable to Stage, and appears to scale approximately linearly. Population sizes of 1 to 40 robots are reported. While Swarmbot3d is clearly a powerful tool, it does not appear to be publicly available.

9.5 Stage Internals for Scaling

Stage's design philosophy has been described previously [31]. The essential design feature that enables scaling is that the per-robot computation required for ray tracing - used for object collision detection and generating range sensor data - is *independent of the robot population size.* As ray tracing is by far the most frequent operation, performed possibly hundreds of times per robot per simulation timestep, we can expect that Stage's overall run-time should grow almost linearly with population size.

Another performance bottleneck in the past was the graphical interface. This has been reimplemented using OpenGL. While the interface has become more sophisticated, now presenting 3-dimensional views, it has also gained a significant performance improvement due to highly optimized libraries, drivers and hardware acceleration.

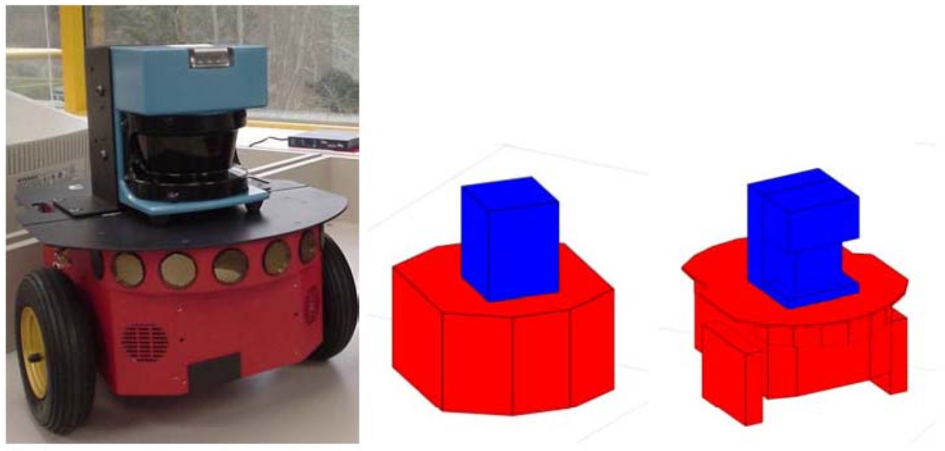
Here we outline how the ray tracing population independence is achieved, and introduce the OpenGL GUI.

9.5.1 Physical object model

Stage models physical bodies as tree assemblies of "blocks". Blocks are specified as arbitrary polygons on the $[x, y]$ plane, extruded into the z axis. The block data structure consists of a pointer to the model that owns the block, an array of two-dimensional points $[[x, y]_0, \ldots, [x, y]_n]$, the array length n, and a $[z^{min}, z^{max}]$ pair indicating the extent of the block in z. This implies that the blocks can be rotated only around the z axis. Because of the missing degrees of freedom, and adopting terminology from computer graphics and video games, we describe Stage as a 2.5D (two-and-a-half dimensional) simulator[13]. Think of blocks as Lego bricks, but with an arbitrary number of side faces.

Block trees can be constructed piecemeal by user code, specified in the worldfile with arrays of points, or loaded from bitmaps in most common file formats (JPEG, PNG, BMP,

[11] `http://www.roboteducation.org/`
[12] `http://www.swarm-bots.org/index.php?main=3&sub=33`
[13] `http://en.wikipedia.org/wiki/2.5D`



(a) Pioneer robot (b) Simple model (c) Complex model

Figure 9.2: Pioneer 3 DX robot from MobileRobots Inc, and two virtual Stage robots, each composed of a StgModelPosition object (in red) and a StgModelLaser object (in blue). The simple model (b) uses 2 blocks described by 16 numbers, and requires 12 rays for collision detection. The complex model (c) uses 11 blocks described by 98 numbers, and requires 74 rays for collision detection. These are our reference models for scaling experiments.

etc.). When interpreting bitmaps, Stage attempts to find a small number of blocks that occupy the same grid as the black pixels in the image.

Figure 9.2 shows two models supplied with Stage, both approximating the popular Pioneer 3DX robot. We compare the speed of Stage simulations using these models below.

9.5.2 Ray tracing

Collisions between blocks and range sensor data are computed using ray tracing. The population of 2.5D blocks is rendered into a 2-dimensional discrete occupancy grid by projecting their shapes onto the ground plane ($z = 0$). Grid cell size is configurable, with a default size of 0.02m. Each grid cell contains a list of pointers to the blocks that have been rendered into that cell. When a block moves, it must be deleted from cells that it no longer occupies and rendered into newly-occupied cells. Non-moving blocks such as walls are rendered into the grid only once.

The ray tracing engine uses well-known techniques, so we summarize it only briefly. For speed and memory efficiency we use a two-level sparse nested grid data structure. Very fast bit-manipulation and integer arithmetic are used to look up grid cell locations from simulation world coordinates specified in meters. Ray tracing involves walking through the nested grid data structure using Cohen's integer line-drawing algorithm [39]. This was found empirically to be faster than the well-known floating-point alternative by Amanantides & Woo [1]. As the ray visits each non-empty cell, we inspect the z-extent of the blocks at that cell, and the properties of the models owning the block, to see if they interact with the ray.

We compared quadtree and kd-tree implementations with our nested grid, and found the grid to run considerably faster, probably due to better memory locality and thus better cache performance. This new ray tracing engine is the main source of Stage's recent performance improvements, and efforts to further improve performance should focus here.

9.5.3 User interface

New in Stage v3 us a user interface using OpenGL [14] and the Fast Light Toolkit framework (FLTK) [15], chosen for speed, ease of use, and wide availability. The new graphics and user interface implementation is the second most significant performance improvement after ray tracing. The OpenGL interface provides a full 3-dimensional view of the world, and alpha blending and antialiasing provide scope for sophisticated effects for visualization of sensor data, motion history, etc.

Figures 9.4, 9.6 and 9.7 show screenshots from Stage demonstrating the new 3-dimensional view. OpenGL takes advantage of graphics processor (GPU) hardware, and we plan to take further advantage of the GPU and also to design more advanced sensor and robot state visualizations (see Future Work section).

9.6 Using Stage as a Library

To date, Stage is most commonly used with Player, to form the Player/Stage system. Robot controllers are written as Player clients, and communicate with Player/Stage through network sockets. It is not well known that Stage version 2 is quite usable as a stand-alone C library, allowing users to embed a complete Stage simulation into their programs. This is done, for example, in MobileSim[16], the simulator shipped by MobileRobots, Inc. to support their range of robots. MobileSim author Reed Hedges contributes bugfixes and features back to the Stage library. Until this paper, we have not publicised this way of using Stage.

However, Player clients send and receive data asynchronously with Player through the host's networking subsystem. This requires significant interaction between the client and server processes and the host operating system, with frequent context switches and likelihood of messages waiting in queues. This overhead may not be acceptable when attempting to scale robot populations. As the Player client-server link is asynchronous, it is not generally possible to run repeatable experiments. Interaction with the OS scheduler and ambient processes means that Player clients are subject to apparently stochastic time delays and will produce stochastic robot behaviour. Some users (including the author) have found it difficult to maintain good enough synchronization between client and Player/Stage v.2 when the host machine is heavily loaded, for example by a large Stage simulation. In particular, the latency experienced by a client is not independent of robot population size, leading to poor robot behaviour. Efforts are underway to solve this problem.

For large scale populations, in cases where the Player features are not required, we now recommended using Stage directly as a library. Stage is deterministic, so a simulation using only deterministic robot controllers will be perfectly repeatable. Sensor data and control commands are contained within a single running process, avoiding costly context switches and exploiting the CPU cache.

Stage version 3 is now a C++ library, which can be used to create a complete simulator by creating a single "StgWorld" object and accessing its methods. Rather than devote a lot of text to this subject, we provide a complete minimal instance of Stage program in Figure

[14]`http://www.opengl.org`
[15]`http://www.fltk.org`
[16]urlhttp://robots.mobilerobots.com/MobileSim

```
#include <stage.hh>

int main( int argc, char* argv[] )
{
  StgWorld::Init( &argc, &argv );  // initialize libstage
  StgWorldGui world( 800, 600, "My Stage simulation");
  world.Load( argv[1] ); // load the world file

  // get a pointer to a mobile robot model
  // char array argv[2] must match a position model named in the worldfile
  StgModelPosition* robot =  (StgModelPosition*)world.GetModel( argv[2] );
  robot->Subscribe();

  world.Start();   // start the simulation clock running

  while( ! world.TestQuit() )
    if( world.RealTimeUpdate() )
    {
       // [ read sensors and decide actions here ]

       robot->Do( forward_speed, side_speed, turn_speed );
    }

  delete robot;
  exit( 0 );
}
```

Figure 9.3: Runnable C++ source code for a minimal but complete Stage robot simulation. Error checking is omitted for brevity. The resulting compiled program could be run with the worldfile of Figure 9.4 like so: `./mystagesim minimal.world MySimpleRobot`. The resulting simulation would look like the screenshot in Figure 9.5.

9.3, a matching world description file and a screenshot of the resulting world in Figure 9.4. This should be sufficient for a programmer to grasp the main ideas. Several examples are provided in the Stage distribution.

9.7 A Benchmark for Swarm Simulation

We seek to obtain empirical run-time data for massively multi-robot simulation systems, in order to evaluate them and compare their performance. Even while data for no other simulator is available, the benchmark would allow Stage users to evaluate the effect of their code additions and optimizations. We have been unable to find a suitable benchmark in the literature, so we propose the following simple procedure. It should be possible to reproduce this procedure very closely on any comparable system.

A benchmark robot controller and world description are required, including a population of robot models. For simplicity, run the identical robot controller concurrently on every robot. Start the simulation and measure the real time it takes to simulate a fixed amount of simulated time. For each [controller, world description] pair, obtain the run-time for

```
# Simulation worlds are specified using a simple tree-structured
# configuration file, or by user programs making individual object
# creation and configuration function calls. This is an example of a
# simple configuration file

# size of the world in meters [x y z]
size [16 16 3]

# create and specify an environment of solid walls loaded from an image
model
(
  # size of the model in meters [x y z] - fill the whole world
  size3 [16 16 0.5]
  color "gray30"

  # draw a 1m grid over this model
  gui_grid 1

  # this model can not be moved around with the mouse
  gui_movemask 0

  # interpret this image file as an occupamcy grid and
  # construct a set of blocks that fill this grid
  bitmap "bitmaps/cave.png"

  # model has a solid boundary wall surrounding the blocks
  boundary 1
)

# create and specify a mobile robot carrying a laser scanner
position
(
 name "MySimpleRobot"
 color "red"
 size3 [0.5 0.5 0.4]

 # pose of the model in the world [x y heading]
 pose [-2.551 -0.281 -39.206]

 laser()
)
```

Figure 9.4: Simple example of a Stage configuration file ("world file"). A screenshot of the world it produces is shown in Figure 9.5

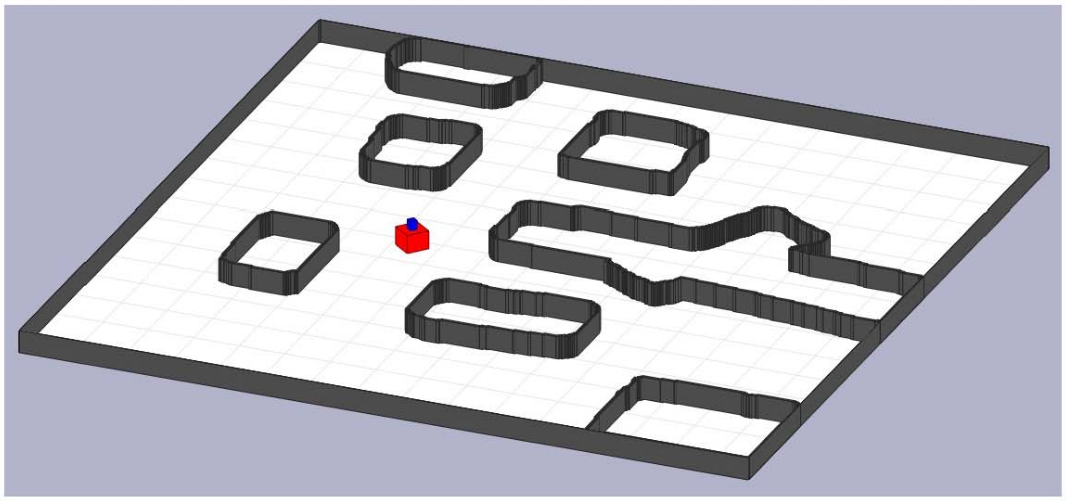


Figure 9.5: Screen shot of the world created by the worldfile shown in Figure 9.4.

a single robot, then a series of exponentially increasing population sizes. For simulators or controllers with stochastic behaviour, repeat each experiment to sample the distribution of run times.

9.7.1 Benchmark robot controller

It is unlikely that any single robot controller will satisfy every user, and any "real" controller that performs a particular useful swarm behaviour from the literature could be argued to be too specialized. We prefer a controller that uses very little computation and memory compared to the simulator, to avoid controller costs masking the simulator performance. Yet we want to see robot behaviour which is somewhat representative of real swarm robotics research.

A canonical application for multi-robot systems, with approaches ranging from swarm intelligence to centralized planning, is wireless sensor network deployment and maintenance ([96, 44, 20] plus several ICRA articles cited above) and the subject of past and present DARPA projects. Multi-robot deployment overlaps with other canonical tasks such as exploration and foraging, though it is not a good model of trail-following or cooperative manipulation.

A minimal version of the deployment problem is dispersal into the environment, which can be achieved using a mobile robot with an array of directional range sensors using the following trivial algorithm:

```
1. gather array of $(range, direction)$ vectors

2. compute vector sum $S$ of array

3. IF( $(heading - \angle S \approx 0$ ) AND ( free space ahead ))
   THEN ( go forwards )
   ELSE ( turn to reduce heading error )

4. GOTO 1
```

With suitable parameter choices depending on robot speed, number of sensors, maximum obstacle detection ranges, etc., this algorithm is highly effective at dispersing the robots until each robot has no object inside its sensor range. The algorithm is a simple approximation of Howard's approach [43], is trivial to implement, uses a single, commonly available class of sensor, and has the advantages of being stateless, thus requiring no per-robot storage between updates, and deterministic, avoiding the need for repeated trials.

Dispersal presents a worst-case scenario for grid-based simulations like Stage, as the amount of free space through which rays must propagate is maximized. Geometric simulations and those that track object bounding boxes may have an easier time, as the frequency of bounding box overlaps is minimized. If it is desirable to test the simulation performance for tightly clustered robots, the controller can trivially be inverted to produce local clustering.

We use this controller for the following benchmarking experiments, and hope that it may be useful for others or inspire (or provoke) the design of a superior method.

9.8 Stage Benchmarks

We present benchmark results for Stage using multiple simulation scenarios. It is impractical to exhaustively explore the huge space of Stage configurations, but our choices span a range intended to be of interest to different types of user. The configurations described here should be straightforward to reproduce on most alternative simulators, and the files used to implement them will be included with the Stage v3 distribution.

9.8.1 Ray Tracing resolution

The spatial resolution of the underlying ray tracing engine was fixed for all tests at 0.02m. This is Stage's default setting, and an informal survey of users suggest that few people ever change this parameter. This means that range sensors and collision detection is accurate to this value. As discussed above, Stage contains no noise models for range data, so ranges are effectively quantized at 0.02m.

9.8.2 Environments

We use two different environments created by the author, included for several years in the Stage distribution and used in papers by multiple researchers: the "cave world" and the "hospital world". The cave was drawn by hand in 1998, and models a moderately constrained environment, either a crude artificial structure or a regular natural structure. The hospital was created in 1999 by hand-editing a CAD drawing of the floorplan of the disused military hospital at Fort Sam Houston, San Antonio, Texas, site of previous DARPA robot demonstrations. The bottom-left hand corner of the hospital world is popularly used as a generic indoor environment, and is referred to as the "hospital section".

Once loaded from their original bitmap files, the cave consists of 489 blocks and is scaled to fill a 16m by 16m world. The hospital consists of 5744 blocks, scaled approximately life size 140m by 60m. All blocks are rectangular polyhedra, with edges aligned with the global axes. The cave and hospital maps are shown in Figure 9.6 and 9.7 respectively.

9.8.3 Mobile robot models

We use the two mobile robot models shown in Figure 9.2, similar but for the number and complexity of the blocks that make up their bodies. They are 0.44m long by 0.38m wide by 0.22m high. Compared to the octagonal "simple" robot model the "complex" robot model requires 6 times as many ray tracing operations to detect collisions (or lack of collisions) at each simulation update. It is also more expensive to draw in the GUI window, though the relative costs are hard to estimate without detailed knowledge of the specific OpenGL and rendering implementation. A third model, the "swarmbot" is used for larger scale experiments, and is similar in shape to the "simple" model, but is smaller, occupying a cube with sides of 0.2m.

9.8.4 Sensor models

The "simple" and "complex" mobile robot models are identically equipped with generic Stage range finder models, 16 sensors distributed around the robot's boundary, each computed by casting a single ray through free-space until a sonar-reflecting object is struck. Their maximum range matches the Pioneer robot's sonar sensors at 5m.

The "swarmbot" has 12 range finders spaced evenly around its body, with a maximum range of 2m. This models a long-range infared range-finder or a short range sonar, such as is typically found on small robots designed for swarm experiments.

In addition, we test some scenarios with a model of a scanning laser rangefinder, parameterized to approximate the popular SICK LMS 200 device, except that it gathers 180 samples over its 180 degree field of view, instead of the 361 samples of the real device. The laser model has a maximum range of 8m: the default on the real SICK. Due to its long range and large number of samples, the laser range model is much more computationally expensive than the sonar/infrared model.

9.8.5 Graphics

To examine the performance impact of the graphics rendering and user interface, the experiments were performed with and without the user interface. With the interface enabled, the window is repainted at the default rate of 100ms. We expect that Stage should run faster with the interface disabled.

9.8.6 Host computer

The host computer was an Apple MacBook Pro, with a 2.33GHz Intel Core 2 Duo processor and 2GB RAM, with a an ATI Radeon X1600 graphics chipset with 256MB VRAM, running Mac OS X 10.5.1. At the time of writing this is a high-end laptop, equivalent to a mid-range desktop PC.

9.8.7 Results

The benchmark robot controller was run in the cave world for all 8 permutations of [model type, laser enabled/disabled and graphics enabled/disabled]. Each permutation was run for 600 seconds with population sizes of 1, 10, 50 and 100 robots and the real-world run time recorded in Table 1. The success of the robot-dispersal algorithm can be seen in Figure 9.6,

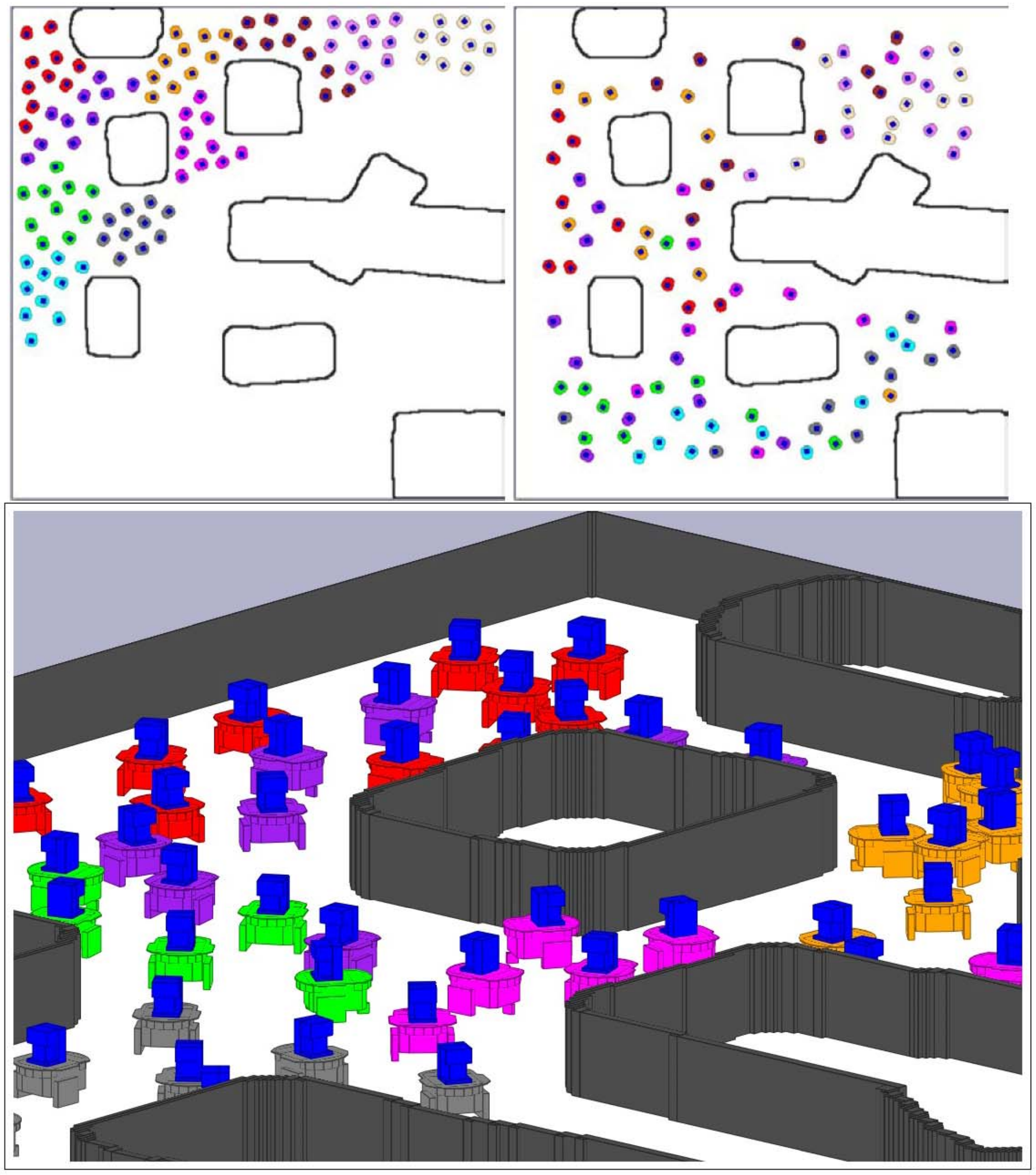


Figure 9.6: The Cave world with 100 complex Pioneer models with lasers in the initial state (top left); after 600 simulated seconds (top right); and a 3D view of the center of the world at 60 seconds (bottom).

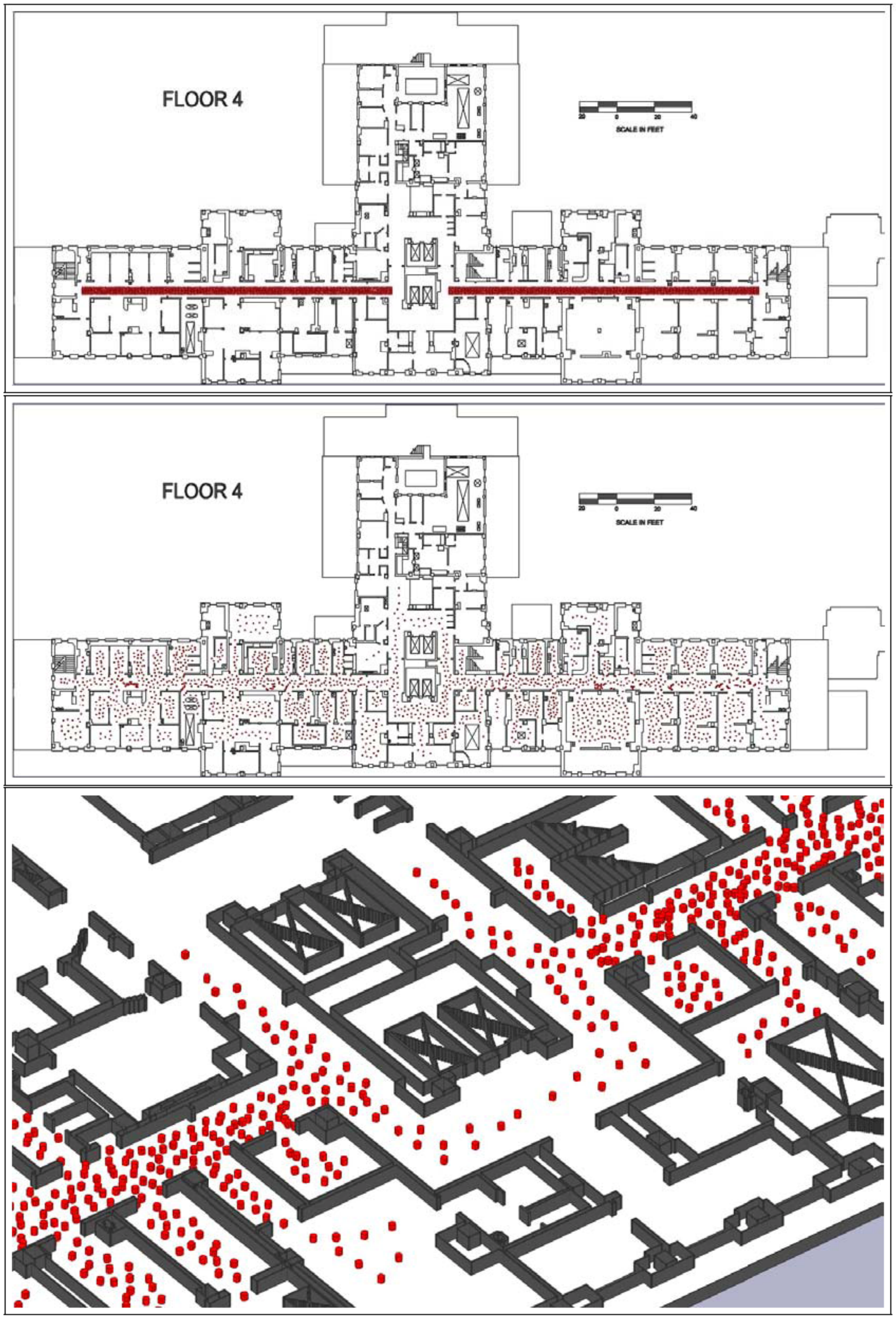


Figure 9.7: The Hospital world with 2000 swarmbots in its initial state (top) (robots are tightly packed and appear as a dark horizontal band); after 600 simulated seconds (middle); and a 3D view of the center of the world at 60 seconds (bottom).

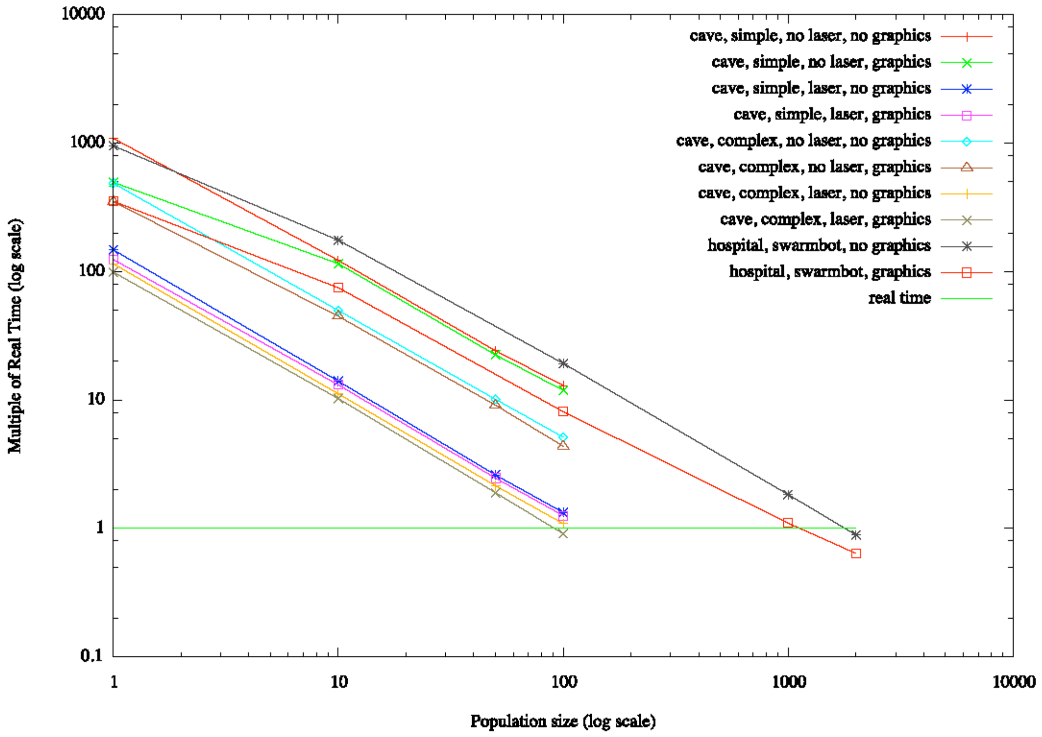


Figure 9.8: Summary of benchmark results. The ratio of simulation time to real time is plotted against population size on a log/log scale, for each world configuration. Anything above the $x = 1$ line is running faster than real time. The run-time performance scales approximately linearly with population size in the range tested.

Model type	Laser	Graphics	Run time for population size			
			1	10	50	100
Simple	-	-	0.55	4.98	24.78	46.23
”	-	yes	1.21	5.2	26.69	50.44
”	yes	-	4.06	42.70	229.48	449.44
”	yes	yes	4.83	45.71	243.28	476.34
Complex	-	-	1.23	12.03	59.15	116.69
”	-	yes	1.71	13.35	65.71	137.15
”	yes	-	5.21	53.61	278.47	550.11
”	yes	yes	6.05	58.44	315.83	659.66

Table 9.1: Cave world Results: real-world run time for 600 seconds of simulation time, for the Cave world in 40 different configurations. Cave world is shown in Figure 9.6. Simple and Complex models are shown in Figure 9.2

Model type	Graphics	Run time for population size				
		1	10	100	1000	2000
Swarmbot	-	0.63	3.41	31/13	325.77	674.00
Swarmbot	yes	1.70	8.00	73.87	546.67	935.41

Table 9.2: Hospital results: real-world run time for 600 seconds of simulation time, for the Hospital world in 40 different configurations. Hospital world and swarmbot model are shown in Figure 9.7.

which shows a global view of the Cave world with 100 complex robots at zero seconds and 600 seconds, and an intermediate 3D view of the center of the world at 60 seconds.

The benchmark robot controller was also run for 600 seconds in the hospital world, with graphics enabled and disabled, for population sizes of 1, 10, 100, 1000 and 2000 robots. The real world run times are recorded in Table 2. The success of the robot-dispersal controller can be seen again in Figure 9.7, where 2000 swarmbots are shown in the hospital world at zero and 600 seconds, and in an intermediate 3D view at 60 seconds.

We see that, as expected, larger population sizes take longer to run. Use of the complex model and the laser scanner also increased run time, as we would expect from their ray tracing demands.

To compare the run time data between trials, we plot the ratio of simulated to real time (i.e. multiples of real time (MRT), or “speed-up factor”) in Figure 9.8. The population size is plotted against MRT on a log/log scale, and similar scenarios, differing only in population size, are connected with lines to form a scaling curve. We see that all the curves are approximately linear, indicating that Stage run time does increase linearly with population size as hoped. We also see that all but the three most demanding simulations ran faster than real time.

To summarize the results, these data show that with a trivial robot controller Stage can simulate small populations of complex robots much faster than real time. Populations of around 100 complex robots or 1,000 simple robots can run in real time or less.

9.9 Future Work

9.9.1 Performance and Scaling

We are interested in running experiments with populations one to two orders of magnitude larger than we have considered in this paper. Populations of 10,000 to 100,000 robots are in the regime of interesting natural swarms. To achieve the required further 10 to 100 times increase in performance, without fundamentally changing our simulation model, we have three main areas of investigation, listed in reverse order of potential scaling benefit:

Optimize code efficiency

There is certainly scope for improvement in the performance of Stage's frequently run inner loops - mainly ray tracing and graphics rendering. In addition, to date little attention has been paid to per-robot memory size and layout. More careful memory use could further improve cache hit rates, and allow larger populations before exceeding a host machine's physical memory.

Concurrency - threads

Multi-core CPUs and multi-CPU machines have become common. In principle, Stage can benefit from parallel computation. Sections of the world that do not interact with each other can run concurrently without any synchronization overhead. Thus we could in principle obtain a near N-times speed increase with N additional processor cores. Currently fast machines with 8 cores are available. The difficulty comes in maintaining suitable partitions, and finding linear time or better algorithms for partitioning physical simulations is an interesting area for research.

Concurrency - cluster computing

The greatest opportunity for massive scaling comes with cluster computing, in which many hosts can be pooled into a distributed system (e.g. Google's huge systems [6]). Distributing Stage over a cluster is a similar problem to using threads, but with much greater between-partition communication costs, and the addition of reliability issues that can usually be ignored on single machines. We will investigate a distributed Stage.

9.9.2 Other plans relevant to swarm robotics

Multi-robot visualization tools

We would like tools for analysing, debugging and presenting the state evolution of large-scale multi-robot systems. We wish to visualize the external and internal state of thousands of agents; to obtain insight into important events, causal relationships, patterns and problems. Exploiting the new OpenGL view of the world, we will gradually add visualizations to the main distribution. For example, Stage-3.0.0 provides various views of the robot's movement history.

Standard API

Stage currently offers a custom API for creating simulations and writing robot controllers. It would be more useful if user's robot controllers were directly portable to Player, and other robot interfaces or simulators. The Player Project has begun work on creating a portable robot controller API, based on the Player Abstract Device Interface, to be implemented initially in Player and Stage. This is a long term community effort.

Portability

Stage v3 is designed to be portable. It currently runs on all major systems except Windows. We aim to have Stage build and run on Windows as standard by the middle of 2008. The main goal of this effort is to make Stage more accessible for students.

9.10 Conclusion

We have argued that Stage may be useful to swarm robotics researchers because (i) it supports large numbers of robots at real time or better; and (ii) it is well known in the multi-robot systems community, and therefore has methodological advantages compared to a dedicated simulator.

To demonstrate the new performance of Stage version 3, we presented and used a simple benchmark for mobile robot simulators. Stage's run time was shown to scale approximately linearly with population size up to 100 computationally demanding robots or 2,000 simple robots.

Stage is not useful for every project, but version 2 has proved itself useful to many. Stage version 3 adds a significant performance increase to this open, free and well-known platform.

9.11 Supporting files

All source code and configuration files required to reproduce these benchmarks are included in Stage source code distributions since Stage-3.0.0, in the directory `worlds/benchmark`. The configuration files are either human-readable text files or PNG format bitmaps. These files and this paper should be sufficient information to adapt the benchmark for another simulator.

9.12 Acknowledgements

Thanks to Brian Gerkey, Reed Hedges, Toby Collett, Radu Rogan Rusu, Pooya Karimian and the Player Project development community. Thanks to Andrew Howard for his work on earlier versions of Stage.

Bibliography

[1] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *Eurographics '87*, pages 3–10. Elsevier Science Publishers, Amsterdam, North-Holland, 1987.

[2] Monica Anderson, Laurence Thaete, and Nathan Wiegand. Player/stage: A unifying paradigm to improve robotics education delivery. In *Workshop on Research in Robots for Education at Robotics: Science and Systems Conference*, 2007.

[3] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. A distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Trans. Robot. Autom.*, 15(5):818–828, 1999.

[4] D. Austin, L. Fletcher, and A. Zelinsky. Mobile robotics in the long term - exploring the fourth dimension. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 613–618, Wailwa, Hawaii, October 2001.

[5] Tucker Balch. *Behavioral Diversity in Learning Robot Teams*. PhD thesis, College of Computing, Georgia Institute of Technology, December 1998.

[6] L. A. Barroso, J. Dean, and U. Holzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[7] M.A. Batalin and G.S. Sukhatme. The analysis of an efficient algorithm for robot coverage and exploration based on sensor network deployment. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3478–3485, 18-22 April 2005.

[8] Maxim A. Batalin and Gaurav S. Sukhatme. Efficient exploration without localization. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2714–2719, Taipei, Taiwan, 2003. IEEE.

[9] Maxim A. Batalin and Gaurav S. Sukhatme. Using a sensor network for distributed multi-robot task allocation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 158–164, New Orleans, LA, USA, April 26 - May 1, 2004.

[10] Gérard M. Baudet. Asynchronous iterative methods for multiprocessors. *J. ACM*, 25(2):226–244, 1978.

[11] D. P. Bertsekas. Distributed dynamic programming. *IEEE Trans. Autom. Control*, 27(3):610–616, 1982.

[12] Andreas Birk. Learning to survive. In *Procs. of the 5th European Workshop on Learning Robots*, pages 1–8, Bari, Italy, 1996.

[13] C. Boutilier and R. Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.

[14] Michael Brenner. Multiagent planning with partially ordered temporal plans. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1513–1514, Acapulco, Mexico, 2003.

[15] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1&2):3–15, June 1990.

[16] Sarah Brown, Mauricio Zuluaga, Yinan Zhang, and Richard Vaughan. Rational aggressive behaviour reduces interference in a mobile robot team. In *Proceedings of the International Conference on Advanced Robotics (ICAR)*, Seattle, Washington, July 2005.

[17] Mark Busch, Marjorie Skubic, James Keller, and Kevin Stone. A robot in a water maze: Learning a spatial memory task. In *Proceedings of the IEEE International Conference on Robotics and Automation*, April 2007.

[18] Stefano Carpin, Mike Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. Usarsim: a robot simulator for research and education. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Rome, Italy, 2007.

[19] H.J. Chang, C.S.G. Lee, Y. Lu, and Y.C. Hu. Simultaneous localization and mapping with environmental structure prediction. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4069–4074, May 15-19, 2006.

[20] R. S. Chang and S. H. Wang. Self-deployment by density control in sensor networks. *IEEE Transactions on Vehicular Technology*, to appear 2007.

[21] J. Cortes, S. Martinez, and F. Bullo. Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions. *IEEE Trans. Autom. Control*, 51(8):1289–1298, 2006.

[22] Jeff Craighead, Robin Murphy, Jenny Burke, and Brian Goldiez. A survey of commercial & open source unmanned vehicle simulators. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Rome, Italy, 2007.

[23] Minh Binh Do and Subbarao Kambhampati. Sapa: A multi-objective metric temporal planner. *J. Artif. Intell. Res. (JAIR)*, 20:155–194, 2003.

[24] Gregory Dudek and Nicholas Roy. Multi-robot rendezvous in unknown environments, or, what to do when you're lost at the zoo. In *Proceedings of the AAAI National Conference Workshop on Online Search*, Providence, Rhode Island, July 1997.

[25] Richard E. Fikes and Nils J. Nilsson. STRIPS, a retrospective. In *Artificial intelligence in perspective*, pages 227–232, Cambridge, MA, USA, 1994. MIT Press.

[26] M. S. Fontán and M. J Matarić. Territorial multi-robot task division. *IEEE Transactions on Robotics and Automation*, 14(5), 1998.

[27] M. Fox and D. Long. Pddl2.1: An extension of pddl for expressing temporal planning domains. *Journal of AI Research*, 20:61–124, 2003.

[28] Mark Fox. ISIS: A retrospective. In M.Zweben and M. Fox, editors, *Intelligent Scheduling*, San Francisco, 1994. Morgan Kaufmann Publishers.

[29] Jakob Fredslund and Maja J. Mataric. Huey, dewey, louie, and gui - commanding robot formations. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2002, May 11-15, 2002, Washington, DC, USA*, pages 175–180. IEEE, 2002.

[30] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan. ASPEN: A framework for automated planning and scheduling of spacecraft control and operations. In *Proc. International Symposium on AI, Robotics and Automation in Space*, 1997.

[31] Brian Gerkey, Richard T. Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics (ICAR)*, Coimbra, Portugal, June 2003.

[32] Brian P. Gerkey. *On Multi-Robot Task Allocation.* PhD thesis, University of Southern California, August 2003.

[33] Brian P. Gerkey and Maja J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *I. J. Robotic Res.*, 23(9):939–954, 2004.

[34] Brian P. Gerkey, Richard T. Vaughan, Kasper Støy, Andrew Howard, Gaurav Sukhatme, and Maja J. Matarić. Most valuable player: A robot device server for distributed control. In *Proceeding of the IEEE/RSJ International Conference on Intelligent Robotic Systems (IROS)*, Wailea, Hawaii, November 2001. IEEE.

[35] D. Goldberg and M.J. Matarić. Maximizing Reward in a Non-Stationary Mobile Robot Environment. *Autonomous Agents and Multi-Agent Systems*, 6(3):287–316, 2003.

[36] S. Gueron and R. Tessler. The fermat-steiner problem. *The American Mathematical Monthly*, 109(5):55–129, May 2002.

[37] K. Halsey, D. Long, and M. Fox. CRIKEY - a planner looking at the integration of scheduling and planning. In *Proceedings of the Workshop on Integration Scheduling Into Planning at 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, pages 46–52, June 2004.

[38] Heiko Hamann and Heinz Wörn. Embodied computation. *Parallel Processing Letters*, 17(3):287 – 298, September 2007.

[39] Paul Heckbert, editor. *Graphics Gems IV.* Academic Press, 1994.

[40] A. Hedenström. Optimal migration strategies in animals that run: a range equation and its consequences. *Animal Behavior*, 66:631–636, 2003.

[41] B. Holldopler and E.O. Wilson. *The Ants.* Springer-Verlag, 1990.

[42] Alasdair I. Houston and John M. McNamara. *Models of Adaptive Behaviour.* Cambridge University Press, 1999. SFU library QL 751.65 M3 H687.

[43] Andrew Howard, Maja J. Matarić, and Gaurav S. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems*, pages 299–308, 2002.

[44] Andrew Howard, Lynne E. Parker, and Gaurav S. Sukhatme. Experiments with large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *International Journal of Robotics Research*, 25(5):431–447, May 2006.

[45] A.S. Morse J. Lin and B. D. O. Anderson. The multi-agent rendezvous problem. In *Proceedings of 42th IEEE Conf. Decision and Control*, pages 1508–1513, Maui,Hawaii, 2003.

[46] A. Jones and J. Rabelo. Survey of job shop scheduling techniques, 1998.

[47] Ari K. Jónsson, Paul H. Morris, Nicola Muscettola, Kanna Rajan, and Benjamin D. Smith. Planning in interplanetary space: Theory and practice. In *Proc. Interntational Conference on Artificial Intelligence Planning Systems*, pages 177–186, 2000.

[48] Alex Kacelnik and Melissa Bateson. Risky theories - the effects of variance on foraging decisions. *American Zoologist*, 36(4):402–434, 1996.

[49] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS*, pages 2149–2154, Sendai, Japan, September 2004.

[50] H.T. Kung. Synchronized and asynchronous parallel algorithms for multiprocessors. In *Algorithms and Complexity*, pages 153–200. New York:Academic, 1976.

[51] Y. Kupitz and H. Martini. Geometric aspects of the generalized Fermat-Torricelli problem. *Bolyai Society Mathematical Studies*, 6:55–129, 1997.

[52] Mark A. Lanthier, Doron Nussbaum, and Tsuo-Jung Wang. Calculating the meeting point of scattered robots on weighted terrain surfaces. In *CATS '05: Proceedings of the 2005 Australasian symposium on Theory of computing*, pages 107–118, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[53] K. Lerman and A. Galstyan. Mathematical Model of Foraging in a Group of Robots: Effect of Interference. *Autonomous Robots*, 13(2):127–141, 2002.

[54] Kristina Lerman, Chris Jones, Aram Galstyan, and Maja J Mataríc. Analysis of dynamic task allocation in multi-robot systems. *Int. J. Rob. Res.*, 25(3):225–241, 2006.

[55] Liu Lin and Zhiqiang Zheng. Combinatorial bids based multi-robot task allocation method. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1145–1150, 18-22 April 2005.

[56] Yaroslav Litus, Richard Vaughan, and P. Zebrowski. A distributed heuristic for energy-efficient multi-robot multi-place rendezvous. Submitted for publication, 2008.

[57] Yaroslav Litus, Richard T. Vaughan, and Pawel Zebrowski. The frugal feeding problem: Energy-efficient, multi-robot, multi-place rendezvous. In *Proceedings of the IEEE International Conference on Robotics and Automation*, April 2007.

[58] S.G. Loizou and V. Kumar. Biologically inspired bearing-only navigation and tracking. In *Proceedings of 46th IEEE Conference on Decision and Control*, New Orleans, USA, 2007.

[59] David McFarland and Emmet Spier. Basic cycles, utility and opportunism in self-sufficient robots. *Robotics and Autonomous Systems*, 20:179–190, June 1997.

[60] Olvier Michel. Webots: a powerful realistic mobile robots simulator. In *Proceeding of the Second International Workshop on RoboCup*. Springer-Verlag, 1998.

[61] J. Minguez and L. Montano. Nearness diagram navigation (nd): A new real time collision avoidance approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)*, pages pg. 21–26, Takamatsu, Japan., 2000.

[62] F. Mondada, G. C. Pettinaro, A. Guignard, I. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L.M. Gambardella, and M. Dorigo. SWARM-BOT: a New Distributed Robotic Concept. *Autonomous Robots, special Issue on Swarm Robotics*, 17(2-3):193–221, 2004. September - November 2004 Sponsor: swarm-bots, OFES 01-0012-1.

[63] E.H. Østergaard, G.S. Sukhatme, and M.J. Matarić. Emergent bucket brigading: a simple mechanisms for improving performance in multi-robot constrained-space foraging tasks. *Proceedings of the fifth international conference on Autonomous agents*, pages 29–30, 2001.

[64] David Payton, Mike Daily, Regina Estowski, Mike Howard, and Craig Lee. Pheromone robotics. *Auton. Robots*, 11(3):319–324, 2001.

[65] Rolf Pfeifer. Building fungus eaters: Design principles of autonomous agents. In *From Animals to Animats, Cambridge, MA: MIT Press*, volume 4, 1996.

[66] G. Polya. *Mathematics and plausible reasoning, 2 vols.* Princeton, 2 edition, 1968. vol 1. Induction and analogy in mathematics; vol 2. Patterns of plausible inference.

[67] J.B. Rosen and G.-L. Xue. Computational comparison of two algorithms for the euclidean single facility location problem. *ORSA Journal on Computing*, 3(3), 1991.

[68] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, 2003.

[69] P.E. Rybski, A. Larson, H. Veeraraghavan, M. LaPoint, and M. Gini. Communication strategies in Multi-Robot Search and Retrieval: Experiences with MinDART. *Proceedings of the Seventh International Symposium on Distributed Autonomous Robotic Systems (DARS-04)*, pages 301–310, 2004.

[70] K. Schlude. From robotics to facility location: Contraction functions, Weber point, convex core. Technical Report 403, Computer Science, ETHZ, 2003.

[71] C. Scrapper, S. Balakirsky, and E. Messina. Moast and usarsim - a combined framework for the development and testing of autonomous systems. In *Proceedings of the SPIE Defense and Security Symposium*, 2006.

[72] Anil K. Seth. The ecology of action selection: Insights from artificial life. *Philos. Trans. R. Soc. B*, April 2007.

[73] D.A. Shell and M.J. Mataric. Directional audio beacon deployment: an assistive multi-robot application. *Proceedings of the IEEE International Conference on Robotics and Automation,*, 3:2588–2594, 26 April-1 May 2004.

[74] Dylan A. Shell and Maja J. Matarić. On foraging strategies for large-scale multi-robot systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2717–2723, Beijing, China., Oct 2006.

[75] Milo Silverman, Dan M. Nies, Boyoon Jung, and Gaurav S. Sukhatme. Staying alive: A docking station for autonomous robot recharging. In *IEEE International Conference on Robotics and Automation*, pages 1050–1055, Washington D.C., May 2002.

[76] Milo Silverman, Dan M. Nies, Boyoon Jung, and Gaurav S. Sukhatme. Staying alive: A docking station for autonomous robot recharging. In *IEEE International Conference on Robotics and Automation*, pages 1050–1055, Washington D.C., May 2002.

[77] S.L. Smith, M.E. Broucke, and B.A. Francis. Curve shortening and the rendezvous problem for mobile autonomous robots. *IEEE Trans. Autom. Control*, September 2005. Under revision.

[78] Emmet Spier and David McFarland. Possibly optimal decision-making under self-sufficiency and autonomy. *Journal of Theoretical Biology*, 189(3):317–331, December 1997.

[79] David W. Stephens, Joel S. Brown, and Ronald C. Ydenberg. *Foraging.* University of Chicago Press, 2007.

[80] David W. Stephens and John R. Krebs. *Foraging Theory.* Princton University Press, 1986.

[81] Russell J. Stuart and Norvig Peter. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2003.

[82] Herbert G. Tanner and Amit Kumar. Formation stabilization of multiple agents using decentralized navigation functions. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[83] Masanao Toda. *Man, Robot and Society.* Martinus Nijhoff Publishing, 1982.

[84] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Trans. Autom. Control*, 31(9):803–812, 1986.

[85] Iwan Ulrich and Johann Borenstein. Vfh+: Reliable obstacle avoidance for fast mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1572–1577, May 1998.

[86] Cao Yu Uny, Fukunaga Alex S., and Kahng Andrew B. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–27, 1997.

[87] Hal R. Varian. *Microeconomic Analysis.* W. W. Norton, 1992.

[88] Richard T. Vaughan, Kasper Støy, Andrew Howard, Gaurav Sukhatme, and Maja J. Matarić. Lost: Localization-space trails for robot teams. *IEEE Transactions on Robotics and Autonomous Systems*, 18(5):796–812, 2002.

[89] Richard T. Vaughan, Kasper Støy, Gaurav Sukhatme, and Maja J. Matarić. Exploiting task regularities to transform between reference frames in robot teams. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Washington, DC, May 2002.

[90] Richard T. Vaughan, Kasper Støy, Gaurav S. Sukhatme, and Maja J. Matarić. Go ahead, make my day: Robot conflict resolution by aggressive competition. In *Proceedings of the 6th International Conference on Simulation of Adaptive Behaviour (SAB)*, pages 491–500, Paris, France, August 2000.

[91] Richard T. Vaughan, Kasper Stoy, Gaurav S. Sukhatme, and Maja J. Matarić. Whistling in the dark: cooperative trail following in uncertain localization space. In *Proceedings of the fourth international conference on Autonomous agents*, pages 187–194. ACM Press, 2000.

[92] W. G. Walter. *The Living Brain.* W.W. Norton, New York, 1963.

[93] Jens Wawerla and Richard T. Vaughan. Near-optimal mobile robot recharging with the rate-maximizing forager. In *Proceedings of the European Conference on Artificial Life*, pages 776–785, September 2007.

[94] Gerhard Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* The MIT Press, July 2000.

[95] David E. Wilkins and Karen L. Myers. A multiagent planning architecture. In *Artificial Intelligence Planning Systems*, pages 154–163, 1998.

[96] Alan Winfield. Distributed sensing and data collection via broken ad hoc wireless connected networks of mobile robots. In LE Parker, G Bekey, and J Barhen, editors, *Distributed Autonomous Robotic Systems 4*, pages 273–282. Springer-Verlag, 2000.

[97] Wei Ye, Richard T. Vaughan, Gaurav S. Sukhatme, John Heidemann, Deborah Estrin, and Maja J. Matarić. Evaluating control strategies for wireless-networked robots using an integrated robot and network simulation. In *Proceedings of the IEEE International Conferece on Robotics and Automation (ICRA)*, Seoul, Korea, May 2001.

[98] Pawel Zebrowski. Tanker based robot recharging. Master's thesis, Simon Fraser University, 2007.

[99] Pawel Zebrowski, Yaroslav Litus, and Richard T. Vaughan. Energy efficient robot rendezvous. In *Proceedings of the Fourth Canadian Conference on Computer and Robot Vision*, pages 139–148, May 2007.

[100] Pawel Zebrowski and Richard Vaughan. Recharging robot teams: A tanker approach. In *Proceedings of the International Conference on Advanced Robotics (ICAR)*, pages 803–810, Seattle, Washington, July 2005.

[101] J.S. Zelek. Dynamic discovery and path planning for a mobile robot at a cocktail party. *Robot Motion and Control, 1999. RoMoCo '99. Proceedings of the First Workshop on*, pages 285–290, 1999.

[102] Yinan Zhang and Richard T. Vaughan. Ganging up: Team-based aggression expands the population/performance envelope in a multi-robot system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Orlando, Florida, May 2006.

[103] Mauricio Zuluaga and Richard Vaughan. Reducing spatial interference in robot teams by local-investment aggression. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Edmonton, Alberta, August 2005.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.) Simon Fraser University, School of Computing Science Autonomy Lab, Burnaby, B C Canada, V5A 4S6	2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.) UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Autonomous Sustain and Resupply, Phase 1 Report

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.)

Vaughan, R.; Litus, Y.; Wawerla, J.; Lein, A.

5. DATE OF PUBLICATION (Month and year of publication of document.) October 2008	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.) 88	6b. NO. OF REFS (Total cited in document.) 103

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Contract Report

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Suffield
PO Box 4000, Medicine Hat, AB, Canada T1A 8K6

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.) 12sj02	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.) W7702-07R146-001
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.) DRDC Suffield CR 2009-067	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

This report presents our progress to date on the DRDC Autonomous Sustain and Resupply project. We are using the ASR problem as the key motivating problem for the AI and autonomy research in our lab. The document is structured as follows. First we state our general approach, which is unchanged from the proposal. Then we summarize our progress to date, referencing detailed descriptions of our work provided as chapters of this report. Next we provide the research plan, which is slightly revised from the version in our proposal. Next is the literature review, followed by our formal model of the problem, and then a design for a planning-based solution. The remainder is a series of chapters describing our progress during Phase 1 on various aspects of the ASR problem.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

sustain, resupply, self-maintenance, autonomous robotics, navigation, path-planning, logistics